UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis der Ähnlichkeitsbestimmung stilisierter Bilder

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Martin Schumann

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inf. Rodja Trappe

Koblenz, im August 2008

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Ein breites Forschungsgebiet in der Bildverarbeitung stellt das Erkennen und Verfolgen von Objekten über den zeitlichen Verlauf von Bildsequenzen dar. Dabei soll die Veränderung von Lage und Position eines Objekts von Bild zu Bild mit geeigneten Verfahren berechnet werden. Dieser als Tracking bezeichnete Vorgang findet im Wesentlichen Einsatz in Überwachungssystemen unter anderem zur Kontrolle von Verkehrsbewegungen, in Systemen zur Registrierung der Augenbewegungen (Eyetracker) und in der Robotik zur Durchführung von Selbstlokalisation und Navigation. Weitere Anwendungsgebiete sind das Erfassen von Körperbewegungen für die Filmanimationstechnik[1] oder die Steuerung von Anwendungen der Virtuellen Realität, sowie die Bestimmung der Kamerapose in der Erweiterten Realität[2] (AR).

Im Vordergrund soll hier das Verfahren der Augmented Reality stehen. Man bezeichnet damit die Ergänzung der visuell wahrgenommenen Wirklichkeit durch zusätzliche grafische Informationen. Augmented Reality in der Computergraphik realisiert dies durch rechnergestützte virtuelle Einblendungen grafischer Objekte in von Kameras aufgenommene Echtzeit-Videoströme (video-see-through) oder über spezielle AR-Brillen, die eine direkte Einblendung in das Gesichtsfeld des Betrachters zulassen (optical-see-through). Anwendung findet die AR-Technologie besonders im medizinischen und technischen Bereich. Sie wird vorwiegend in den Phasen der Entwicklung und des Designs sowie zur Montage- und Wartungsunterstützung oder für Lern- und Trainingssituationen eingesetzt. Aber auch mobile Informationssysteme und AR-Spiele eröffnen in jüngster Zeit weitere Forschungsfelder.

Neben der echtzeitfähigen Interaktion mit den eingeblendeten Objekten und entsprechendem haptischen Feedback liegt der Aufgabenschwerpunkt von Augmented Reality Systemen in der Berechnung von Lage und Orientierung der bildgebenden Kamera im Raum. Sie stellt den Sichtpunkt des Betrachters dar und ist daher Voraussetzung für das korrekte, mit der Realität deckungsgleiche Rendern der einzublendenden Objekte. Bei gängigen AR-Konzepten wie etwa dem ARToolKit wird das Ermitteln der benötigten Informationen über die Kamera durch das Tracking, dem Finden und Verfolgen von vordefinierten Markern im Bild ermöglicht. Das AR-System liest dazu von einer angeschlossenen Web- oder Videokamera alle aufgenommenen Bilder aus und durchsucht sie nach ihm bekannten Markern. Wird ein solcher Marker im aktuellen Frame gefunden, führt das System

[1] Motion Tracking oder Motion Capture
[2] Augmented Reality (AR)

Berechnungen durch, welche die Position und Orientierung der Kamera relativ zum Marker ergeben. Das virtuelle Objekt kann nun aus Sicht der Kamera gerendert und anhand der gewonnenen 3D-Koordinaten durch Rotation und Translation an der Position des Markers eingeblendet werden.

Es ist jedoch nicht immer praktikabel, Marker in der Umgebung zu platzieren, etwa wenn sie das Allgemeinbild stören würden. Aktuelles Forschungsgebiet ist daher markerloses Tracking, welches immer noch eine große Herausforderung darstellt. Bestehende Verfahren suchen Korrespondenzen von zuvor extrahierten Merkmalen[3] in Bilderfolgen, um aus ihrer Veränderung über die Zeit die Pose der Kamera zu berechnen. Dieses Vorgehen ist nicht immer ganz problemlos und leidet unter Fehlern wie Drift der Kamerapose. Der neue Ansatz der Analyse durch Synthese soll es im Gegensatz dazu ermöglichen, bei gegebener Geometrie der zu trackenden Szene durch Rendering eine optimale Vergleichsgrundlage zum Kamerabild zu schaffen. Wird das Objekt aus mehreren bekannten Kameraposen gerendert und ein Abgleich des Kamerabildes mit diesen synthetisch erzeugten Bildern durchgeführt, dann gibt das gerenderte Bild mit der größten Ähnlichkeit die aktuelle reale Kamerapose wieder. Zu klärende Aspekte betreffen den für das Tracking benötigten Detailgrad des Renderings und die Wahl der zum Vergleich mit dem Kamerabild geeignete Ähnlichkeitsmaße.

Ziel dieser Arbeit soll es sein, markerloses Tracking unter dem Ansatz der Analyse durch Synthese zu realisieren und dabei auf den Einsatz merkmalsbasierter Verfahren zu verzichten. Es werden Kombinationen von Ähnlichkeitsmaßen und Visualisierungen untersucht, um eine bestmögliche Vergleichbarkeit von Bildern zu erreichen. Kamerabild und synthetisches Bild sollen durch den Einsatz von Stilisierungstechniken so verändert und angeglichen werden, dass zu einem gegebenen Kamerabild aus einer Auswahl von gerenderten Bildern jenes erkannt werden kann, welches die reale Kamerapose am exaktesten wiedergibt. Des Weiteren soll untersucht werden, welchen qualitativen Anforderungen das Rendering genügen muss, um möglichst robust gegen Trackingfehler zu sein.

Die Arbeit baut auf der vorbereitenden Studienarbeit [Sch07] auf, welche die Grundlagen für die verwendeten Stilisierungsarten liefert und der die eingesetzten Shader teils in überarbeiteter Form entnommen sind.

1.2 Aufbau

Zunächst soll Kapitel 2 den Einstieg in das Verfahren der Analyse durch Synthese liefern, das den Kern des Trackings darstellt. Es werden vorhandene Methoden aufgezeigt sowie der aktuelle Ansatz beschrieben. In Kapitel 3 wird der gewählte Optimierungsansatz hergeleitet und erläutert. Ka-

[3] Feature Based Tracking

pitel 4 stellt Ähnlichkeitsmaße vor, die sich zum Vergleich von Bildern eignen und gibt eine Übersicht der Anwendungsbereiche. Kapitel 5 geht auf die Stilisierungsformen zur Angleichung gerenderter Bilder und Kamerabilder ein. In Kapitel 6 wird die konkrete Umsetzung des Trackingverfahrens beschrieben. Mit der Analyse der gewonnenen Resultate befasst sich der Schwerpunkt von Kapitel 7 und zeigt des Weiteren die Einsatzfähigkeit auf. Abschließend wird in Kapitel 8 ein Fazit aus der vorliegenden Arbeit gezogen und ein Ausblick auf spätere Arbeiten und deren Verbesserungsmöglichkeiten gegeben.

2 Analyse durch Synthese

2.1 Definition

Weit verbreitet ist der Trackingansatz, Merkmale über eine Sequenz von Kamerabildern zu verfolgen. Merkmale werden in zwei aufeinander folgenden Bildern extrahiert und ihre Korrespondenzen in beiden bestimmt. Aus der Veränderung der Merkmale über die Zeit wird dann die Kamerapose geschätzt. Bei dieser Vorgehensweise ergibt sich jedoch ein nicht unerhebliches Problem. Das Erkennen der Merkmale in zwei Bildern, die zum Vergleich herangezogen werden, ist den Widrigkeiten der Beleuchtung und anderen Bildstörungen unterworfen. Daher kann es zu Korrespondenzproblemen während des Trackingvorgangs kommen, wenn die Merkmale im nächsten Bild nicht eindeutig wiederzuerkennen sind. So kann die Kamerapose verloren gehen, etwa wenn das zu trackende Objekt in einen Schatten eintaucht.

Ein weiterer Nachteil des Trackings über Bildsequenzen ist, dass Fehler Einfluss auf alle folgenden Ergebnisse haben und sich von Bild zu Bild summieren, was zu einem Drift der Kamerapose führen kann. Statt rein bildbasiert zu arbeiten und zwei suboptimale Eingangsbilder zu vergleichen wäre es daher wünschenswert, einen störungsfreien Prototypen als Vergleichsreferenz für jedes einzelne Kamerabild heranzuziehen, damit das Wissen auf einer Seite des Vergleichs als bekannt vorausgesetzt werden kann und sich Fehler reduzieren lassen.

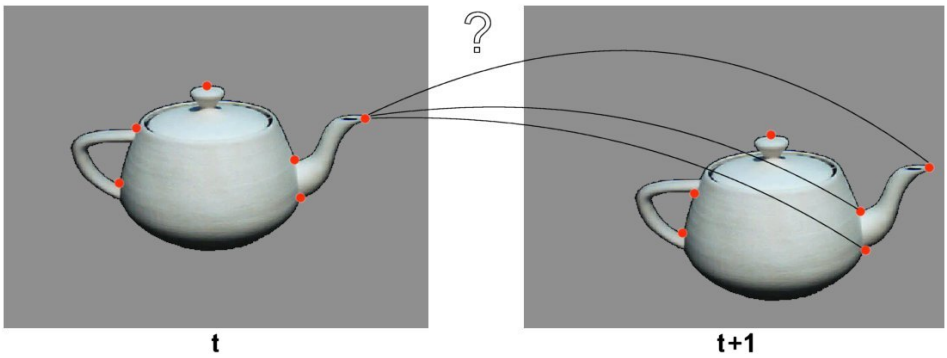

Abbildung 1: Korrespondenzproblem

Erreicht werden kann dies mit Hilfe der Methode der Analyse durch Synthese. Sie wird in [Lie89] beschrieben als Verfahren zum optimierten Vergleich von Objekten mit vorgegebenen Modellen (sog. Modellvergleichsverfahren) und zählt zu den wissensbasierten Bildanalysestrategien. Der Vorteil in der Anwendung eines modellbasierten Verfahrens liegt in dem a priori Wissen, welches das für den Vergleich herangezogene Modell beinhaltet. Mit Hilfe des Modells können Bildprototypen unter verschiedenar-

tigen Bedingungen generiert werden, wobei die Parameter, welche zur Entstehung des synthetischen Bildes führen, bekannt sind und eine Grundlage zur Optimierung der Darstellung liefern. Diese in die Bildentstehung einfließenden Parameter können das Modell selbst betreffen, die Beleuchtung, Kameraposition und Orientierung, Perspektive, Abstraktionstufe und Oberflächeneigenschaften wie etwa Reflektionsgrad oder Textur.

Ein geeignetes Ähnlichkeitsmaß soll dann die Übereinstimmung von generiertem Prototyp und Bildobjekt testen. Je nach Abweichung des Prototypen von dem Eingangsbild werden die Erzeugungsparameter korrigiert und die Darstellung damit weiter optimiert. Die korrigierten Parameter dienen dann zur Erzeugung der folgenden Prototypen. Ziel bei diesem Verfahren ist es, sich mit dem synthetischen Objekt dem realen Objekt so weit wie möglich anzunähern, um dann die durch Optimierung der Synthese gewonnenen Parameter als gültig für die reale Situation anzuerkennen. In [Lie89] wird insbesondere auf die Notwendigkeit hingewiesen, dass die in den Vergleich eingehenden Repräsentationen von synthetischem Prototyp und real aufgenommenem Objekt gleichartig sein müssen, um ein geeignetes Ähnlichkeitsmaß zu definieren. Auch [Moe99] betont, dass sich synthetische und reale Daten auf einem Abstraktionslevel befinden müssen.

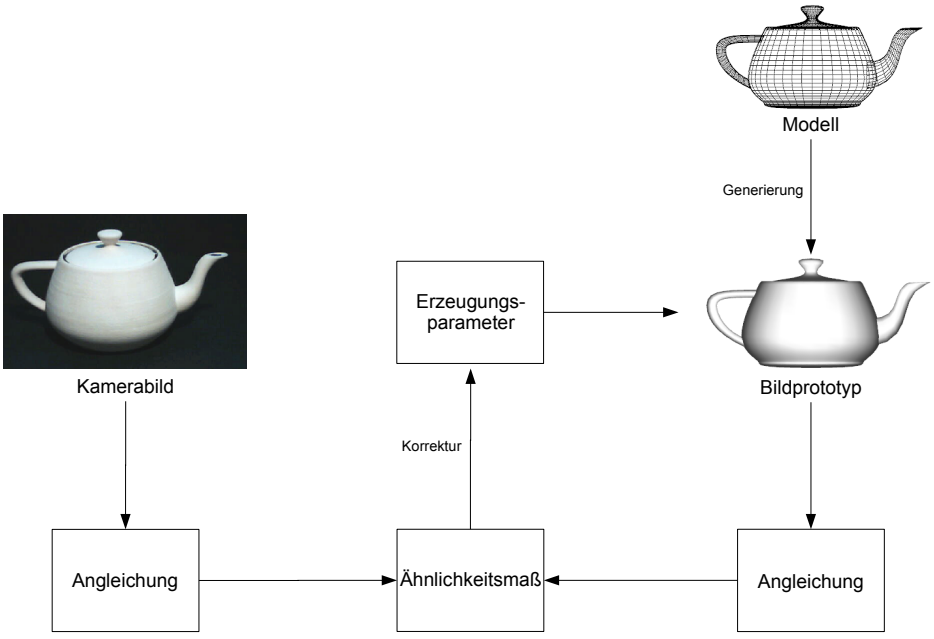

Abbildung 2: Analyse durch Synthese

2.2 Anwendung

Die Anwendung des Prinzips der Analyse durch Synthese findet sich bereits bei [Bel61] aus dem Jahre 1961. Dort wird ein System beschrieben, das zur Sprachverarbeitung dient. Es soll eine minimale Menge von Merkmalen finden, mit denen sich das Frequenzspektrum eines Eingabelautes eindeutig und datenreduziert beschreiben lässt. Das dort vorgestellte System besitzt eine Syntheseeinheit, der Parameter in Form bekannter Lautmerkmale übergeben werden und die daraus synthetische Laute erzeugt. Die Analyseeinheit berechnet daraufhin den Fehler zwischen dem synthetisch erzeugten Laut und einem Eingangssignal. Die Parameter für die Generierung des synthetischen Lautes werden solange angepasst, bis der gemessene Fehler gering wird und sich mit den gefundenen Merkmalen der Eingabelaut möglichst genau reproduzieren lässt.

Im Bereich der Computergraphik und Bildverarbeitung ist die Anwendungsfülle bereits breit gefächert. In [Moe99] wird ein Überblick über den Einsatz von Analyse durch Synthese Techniken im Bereich des Motion Capturing menschlicher Bewegungen gegeben. Das Erkennen von Bewegungen soll mit Hilfe synthetischer Bilder ermöglicht werden. Das Körpermodell eines Menschen ist vorgegeben und wird mit einer Bewegung animiert. Das daraus erzeugte synthetische Bild wird mit dem Kamerabild verglichen und die Bewegung des Modells entsprechend der Abweichung von der realen Bewegung korrigiert bis eine Übereinstimmung gefunden ist.

In [Ton94] bedient man sich der Analyse durch Synthese, um die Modellierung von 3D Objekten zu optimieren. Zunächst wird aus einer Folge von Stereobildpaaren unter Bestimmung der Tiefeninformationen durch Disparität ein grobes 3D Modell erzeugt. Ausserdem wird eine Wissensbasis mit symbolischer Beschreibung des Modells angelegt, die die Polygone zu größeren Einheiten des Modells zusammenfasst. Von diesem Modell werden unter Berücksichtigung der Wissensbasis synthetische Bilder erzeugt und mit den weiteren Kamerabildern verglichen. Abweichungen führen zur Adaption der Struktur des Modells bezüglich der Polygonzüge und ihrer Zuordnung zu den Elementen des Modells, um eine fehlerhafte und unvollständige Modellierung zu verbessern.

Wuest und Stricker [Wue07] setzen modellbasierte Analyse durch Synthese ein, um echtzeitfähiges Tracking der Kamerapose zu ermöglichen. Von einem mit bekannter Kamerapose gerenderten Szenenmodell werden dessen Linienzüge durch die Extraktion der Modellkonturen erstellt. Für den Analyseschritt werden 3D Kontrollpunkte auf den Konturen definiert, welche in das Kamerabild projiziert werden. Es erfolgt eine Suche nach 2D/3D Korrespondenzen, indem für jeden projizierten Kontrollpunkt auf der synthetischen Kontur eine senkrechte Suche in der Umgebung nach

Kanten im Kamerabild durchgeführt wird. Durch Minimierung des Fehlers zwischen dem projizierten 3D Linienmodell und den gefundenen 2D Kanten im Kamerabild wird die Kamerapose für den Vergleich mit dem nächsten Bild bestimmt und der realen Kamerapose angenähert. Ein ebenfalls modellbasierter Trackingansatz, der aus einem Modell generierte Linien- und Punktmerkmale mit dem Kamerabild vergleicht, findet sich in der Diplomarbeit [Ewe06].

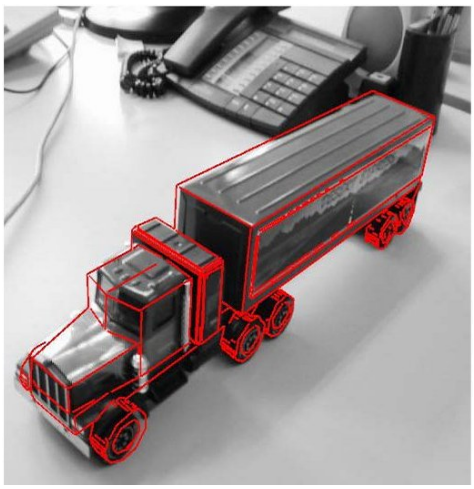

Abbildung 3: Modellbasiertes Linientracking aus [Wue07]

Abbildung 4: Modellbasiertes Linientracking aus [Ewe06]

Das in der Diplomarbeit [Den07] entwickelte System nutzt 2D/3D-Bildregistrierung um die Kamerapose, welche von einem Grobtrackingsystem (GPS) bestimmt wurde weiter zu verfeinern. Eine erste Ansicht des 3D Modells wird aus der von einem GPS bestimmten Grobposition gerendert. Bildabstandsmaße vergleichen gerendertes Bild und Kamerabild. Optimierungsverfahren Schätzen die Pose der Kamera und aus dieser wird eine Folgeansicht generiert. Dies wird iterativ durchgeführt, bis der gemessene Abstand minimal ist.

Achilles [Ach08] entwickelte parallel zu dieser Arbeit einen Ansatz, der Analyse durch Synthese auf Basis von Featuredetektoren nutzt.

Der Kern der vorliegenden Arbeit basiert ebenfalls auf Analyse durch Synthese. Die zu optimierenden Parameter der Bildsynthese betreffen in diesem Fall die Kameraposition und -orientierung, kurz Kamerapose genannt. Sie wird als Posevektor dargestellt, der die Parameter der Translation und Rotation beinhaltet. Die Parameter der Pose werden vom System variiert und mehrere synthetische Bilder aus verschiedenen Ansichten erstellt. Diese werden mit dem Eingangsbild der Kamera verglichen, indem die Ähnlichkeit berechnet wird.

Die Pose des gerenderten Bildes mit der höchsten Ähnlichkeit zum Ka-

merabild wird weiter optimiert, indem das System im Umfeld der letzt besten Pose neue Posen generiert und entsprechende Bilder synthetisiert. Dabei soll die Ähnlichkeit der synthetischen Bilder zum Kamerabild so weit in Übereinstimmung gebracht werden, dass die virtuelle Pose der realen Kamerapose als angenähert gelten kann. Ist M das Modellbild, das mit dem virtuellen Posevektor p_m gerendert wurde und K das Kamerabild mit der zu bestimmenden realen Kamerapose p_k, dann soll p_m variiert werden, bis die Ähnlichkeit A maximal wird:

$$p_m = \underset{p_m}{\operatorname{argmax}} A(M(p_m), K(p_k))$$

$$mit$$

$$p_m, p_k = (t_x, t_y, t_z, r_x, r_y, r_z)$$

$$bis$$

$$M(p_m) = K(p_k) \quad \Rightarrow \quad p_m = p_k$$

Im Unterschied zu den vorherigen Arbeiten wird hier nicht nur auf *einem* synthetisierten Bild gearbeitet, dessen Informationen genutzt werden, um festzustellen welche Veränderung zum nächsten Bild stattgefunden hat. Durch die Synthese mehrerer Bilder werden Hypothesen darüber aufgestellt, wie die Bewegung zwischen Kamera und Objekt für das nächste Bild aussehen könnte, um dann die beste Hypothese als Grundlage für die weitere Posebestimmung zu wählen. Aus Performanzgründen wurde in der Vergangenheit davon abgeraten, den Vergleich auf Bildebene durchzuführen [Lie89]. Stattdessen wurden zum vereinfachten Vergleich Objekt- und Bildmerkmale herangezogen. Entgegen dieser Empfehlung wird in dieser Arbeit, nicht zuletzt aufgrund der Entwicklung schneller und leistungsfähiger Hardware, das Bild als Einheit betrachtet und ein globaler darstellungsbasierter Ansatz zur direkten Ähnlichkeitsbestimmung ohne Merkmale gewählt. Die angesprochene Anforderung der Gleichartigkeit der zu vergleichenden Bilder soll durch die Stilisierung (Kapitel 5) gewährleistet werden.

3 Optimierung

3.1 Analyse des Problems

Nach dem vorgestellten Prinzip der Verwendung von Analyse durch Synthese sollen um eine zuletzt gültige Kamerapose eine bestimmte Anzahl neuer Kameraposen gereriert werden. Unter diesen gilt diejenige als beste Wahl für eine Annäherung an die reale Kamerapose, deren zugehöriges sythetisches Bild die größte Ähnlichkeit mit dem Kamerabild aufweist. Dabei ist die Frage zu klären, wie die neuen Kameraposen genau zu erzeugen sind, um möglichst zuverlässig und mit nicht zu großem rechnerischen Aufwand die maximale Ähnlichkeit zu erreichen. Dies betrifft besonders die Ausdehnung und Dauer der Suche nach dem Maximum der Ähnlichkeit. Um dieses Problem zu lösen bedarf es einer Optimierungsstrategie, die folgendes leisten soll:

> Bestimme den Parameter p aus einer Menge von möglichen Lösungen zur Maximierung des Funktionswertes der Funktion f so, dass für alle übrigen Parameter p' gilt
>
> $$f(p) \geq f(p')$$

Als Nebenbedingung soll gelten, dass so wenige Auswertungen der Funktion benötigt werden sollen wie möglich. Die folgende Analyse des Problems soll das einzusetzende Optimierungsverfahren bestimmen, dessen Bestandteile zu definieren sind:

- Zu optimierende Funktion
- Definitionsmenge des Parameterraums
- Suchraum / Suchstrategie zur Generierung der Parameter
- Startpunkt der Optimierung
- Abbruchkriterium der Suche

Im konkreten Fall ist die zu optimierende **Funktion** das Ähnlichkeitsmaß, das die Übereinstimmung zweier Bilder feststellt. Es handelt sich dabei um ein nichtlineares Optimierungsproblem. Ferner werden zur Optimierung die Funktionswerte des Ähnlichkeitsmaßes direkt herangezogen, was die Anwendung von sogenannten Gradientenverfahren ausschließt, welche die Extrema einer Funktion über die Nullstellensuche der ersten Funktionsableitung finden.

Der **Parameterraum** auf dem operiert wird, lässt sich bestimmen durch den Raum, in dem sich die Kamera bewegen kann. Dieser umfasst 6 Freiheitsgrade in Translation und Rotation, beschrieben durch den Posevektor der Kamera

$$p = (t_x, t_y, t_z, r_x, r_y, r_z)$$

Der Parameterraum ist jedoch kontinuierlich und unendlich, was es unmöglich macht, ihn vollständig zu durchsuchen. Um die Nebenbedingung zu erfüllen, bedarf es daher einer Einschränkung und Diskretisierung auf ein Intervall, in dem die Funktion abgetastet wird. So wird ein **Suchraum** definiert, innerhalb dessen in bestimmten Abständen neue Parameter ausgewählt werden. Zu klären ist an dieser Stelle, ob die Suche durch ein globales oder lokales Optimierungsverfahren erfolgen soll. Hauptproblem der globalen Suche ist die Ineffizienz. Es werden hierbei nicht deterministisch Parameter erzeugt, sondern zufällig generierte Parameter um das letzte Optimum gestreut (Zufallssuche). Dies kann dazu führen, dass die Suche mit einem Schritt sofort mit einem Zufallstreffer auf das Maximum stößt. Es birgt aber auch die Gefahr, immer um das tatsächliche globale Maximum herum Parameter zu generieren ohne es jedoch zu treffen und immer wieder den selben Bereich abzusuchen. Daher ist ein lokales Suchverfahren anzusteben, das in der direkten Nachbarschaft des letzt besten Parameters nach optimaleren Parametern sucht.

Diese Einschränkung des Suchraums wird in Bezug auf das Tracking möglich durch die Annahme kleiner Bewegungen. Wenn davon auszugehen ist, dass sich die Kamera in Translation und Rotation immer nur in kleinen Schritten bewegt, ist es auch nur notwendig einen kleinen Teil des Raumes um die aktuelle Pose zu durchsuchen. So sollte das globale Maximum der Ähnlichkeit schnell zu finden sein, ohne auf ein lokales Maximum zu stoßen. Wie in den grafischen Darstellungen der Ähnlichkeitsmaße in Kapitel 7.1 zu sehen ist, kommt der tatsächliche Verlauf der eingesetzten Ähnlichkeitsmaße dieser Annahme sehr entgegen.

Diese Form der Suche ist deterministisch und regulär, da die Funktion in festen Abständen abgetastet wird. Die Anzahl der zu generierenden Posen ist dabei performanzabhängig zu wählen, denn mit der zu untersuchenden Parameterzahl steigt auch der Aufwand. Die anzuwendende **Suchstrategie** entspricht der Bewegung im Raum. Optimal wird von der aktuellen Pose aus gesehen in alle drei Raumrichtungen entlang der Koordinatenachsen ein neuer Posevektor in positiver wie in negativer Richtung erzeugt. Dasselbe gilt für die Rotation, bei der um jede Koordinatenachse ein Rotationswinkel gleichermaßen positiv und negativ generiert wird. So entstehen 12 Posen, zu denen die entsprechenden Ansichten gerendert werden (Abbildung 5). Dies ist das Minimum an Posen, die ausreichen um alle möglichen Bewegungen abzudecken. Für den Fall, dass keine Bewegung stattfindet, wird zusätzlich die Ansicht der zuletzt besten Pose als 13te gerendert und mit in die Suche einbezogen. Dies verhindert in Ruhelage ein Springen zwischen den Werten (Jittering).

Da sich die Optimierung auch unterschiedlichen Geschwindigkeiten der Bewegung anpassen soll, ist die **Schrittweite** variabel zu wählen. Das durch die generierten Parameter aufgespannte Suchfenster wird dadurch

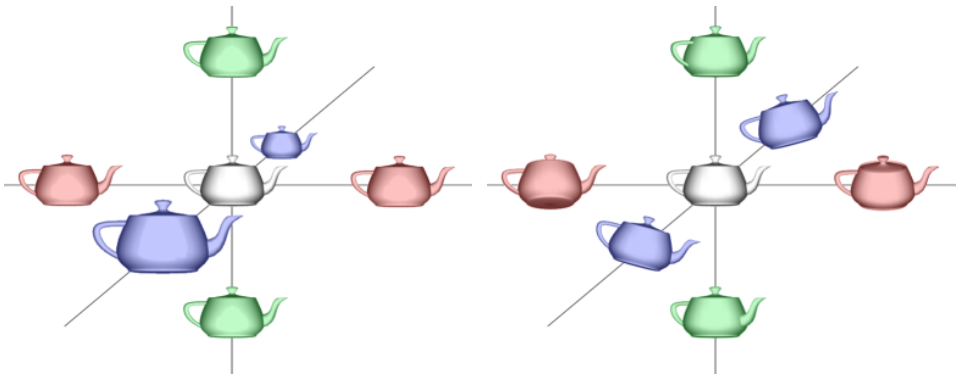

Abbildung 5: Generierung der Ansichten für Translation und Rotation

adaptiv in Abhängigkeit von einem Schwellwert, der bestimmt, wann welche Suchfenstergröße ausgewählt wird. Als Schwellwert dient die gemessene Ähnlichkeit zwischen synthetischem Bild und Kamerabild. Dabei wird die Schrittweite des Suchfensters, in dem die Posen um die zuletzt beste erzeugt werden, dem Funktionswert angepasst. Ist die gemessene Ähnlichkeit hoch, so ist damit zu rechnen, dass sich das Maximum in unmittelbarer Nähe befindet und nur sehr kleine Bewegungen stattfinden. Die Suche wird auf einen engen Raum verfeinert. Sinkt die Ähnlichkeit jedoch stärker ab, was auf eine schnellere Bewegung zwischen Kamera und Objekt hindeutet, so wird die Schrittweite vergrößert und Posen in einer größeren Umgebung gesucht.

Der **Startpunkt** der Optimierung ist aufgrund der lokalen Suche so zu setzen, dass er bereits möglichst nahe am Optimum liegt. Andernfalls erhöht sich neben der Suchzeit auch die Gefahr, dass die Suche auf ein lokales Maximum trifft, da dies der Annahme der kleinen Bewegungen entgegensteht. Die initiale Kamerapose muss also als bekannt vorausgesetzt werden können. Als Alternative zur Vermeidung lokaler Maxima ist ein Verfahren mit mehreren Startpunkten denkbar, sodass verschiedene Lösungswege zum Maximum parallel verfolgt werden. Allerdings geht dies sehr zu Lasten der Performanz. Ein Festfahren der Suche an einem lokalen Maximum bei kleinem Suchfenster kann auch gelöst werden, indem in bestimmten Zeitintervallen ein Suchfenster mit großer Schrittweite eingestreut wird. Dieses Vorgehen soll überprüfen, ob nicht in einer größeren Umgebung noch einen besserer Funktionswert existiert.

Als **Abbruchbedingung** bei einem Optimierungsverfahren kann die Anzahl der Iterationen bei der Suche dienen. Erfolgt über eine bestimmte Zeit lang keine Verbesserung des Funktionswertes mehr, wird die Suche terminiert. Es ist auch möglich einen Schwellwert festzulegen, der zum Abbruch der Suche führt, wenn der Funktionswert oder die Schrittweite bei

der Suche unter diesem liegen. Bei dem aktuellen Ansatz gilt keine Abbruchbedingung. Die Optimierung findet kontinuierlich statt. Auch wenn das globale Maximum erreicht ist, werden weiterhin die umliegenden Posen getestet, um Veränderungen in der Ähnlichkeit festzustellen, die durch Bewegung hervorgerufen werden.

3.2 Best Neighbor Search

Das beschriebene Szenario entspricht dem lokalen Optimierungsverfahren *Best Neighbor Search* [Wei03] unter Verwendung eines adaptiven Suchfensters. Für jeden Optimierungsschritt werden mit einer festen Schrittweite $2n$ Posen um die Pose mit dem aktuell besten Funktionswert erzeugt, wobei n die Anzahl der Freiheitsgrade der zu bestimmenden Parameter ist. Diejenige Pose mit den Parametern, die für die Funktion den besten Wert liefert, wird als Ausgangspunkt für die Generierung der nächsten Posen gesetzt und das Suchfenster anhand des Kriteriums Ähnlichkeit entsprechend angepasst.

Da das Verfahren danach strebt, immer den höchsten Funktionswert zu erreichen wird es auch *Hill-Climbing* genannt. Nach [Aar97] unterscheidet man drei Arten der Suchstrategie bei der lokalen Nachbarschftssuche. *Next Improvement* wählt aus den auszuwertenden Parametern immer den ersten, welcher eine Verbesserung gegenüber dem aktuellen Wert darstellt und springt dann zur nächsten Iteration. *Best Improvement* wertet die Funktion zuerst mit allen generierten Parametern aus, um dann den absolut besten Parameter als beste Lösung zu bestimmen. *Random Neighbor* wählt aus den Parametern des Suchraums einen zufälligen, der dann ausgewertet wird. Die einzig in Frage kommende Variante ist *Best Neighbor*, da jederzeit mit einer Bewegung in alle Richtungen zu rechnen ist. Aus der letzten Bewegungsrichtung wird keine Annahme darüber getroffen, wie sich die Bewegung fortsetzen wird. Somit ist keine Gewichtung oder Auswertungsreihenfolge der generierten Parameter vorzunehmen und alle Parameter sind grundsätzlich gleichwertig zu behandeln. Daher werden auch Parameter ausgewertet, die zuvor bereits einmal in einem anderen Suchfenster getestet wurden, denn es ist auch ein Schritt zurück möglich, wenn sich die Bewegung in die entgegengesetzte Richtung ändert.

In Abbildung 6 ist das Verfahren der Best Neighbor Search grafisch am Beispiel einer als ideal angenommenen Ähnlichkeitsfunktion dargestellt. Zu sehen ist die Aufsicht der Funktionswerte mit dem Maximum in der Mitte des Diagramms. Die Optimierung geschieht hier zur Veranschaulichung für den 2D Fall in einer Ebene mit nur vier neu erzeugten Posen. Die vier Punkte, die durch eine Linie verbunden sind, gehören zum jeweiligen Suchfenster. Rote Punkte entsprechen den Funktionswerten an den Stellen, die durch die generierten Posen erzeugt wurden. Von diesen wird

jeweils eine als optimale durch einen grünen Punkt gekennzeichnet. Die Annäherung an das Maximum geschieht unter Verkleinerung des Suchfensters in dem die neuen Posen generiert werden.

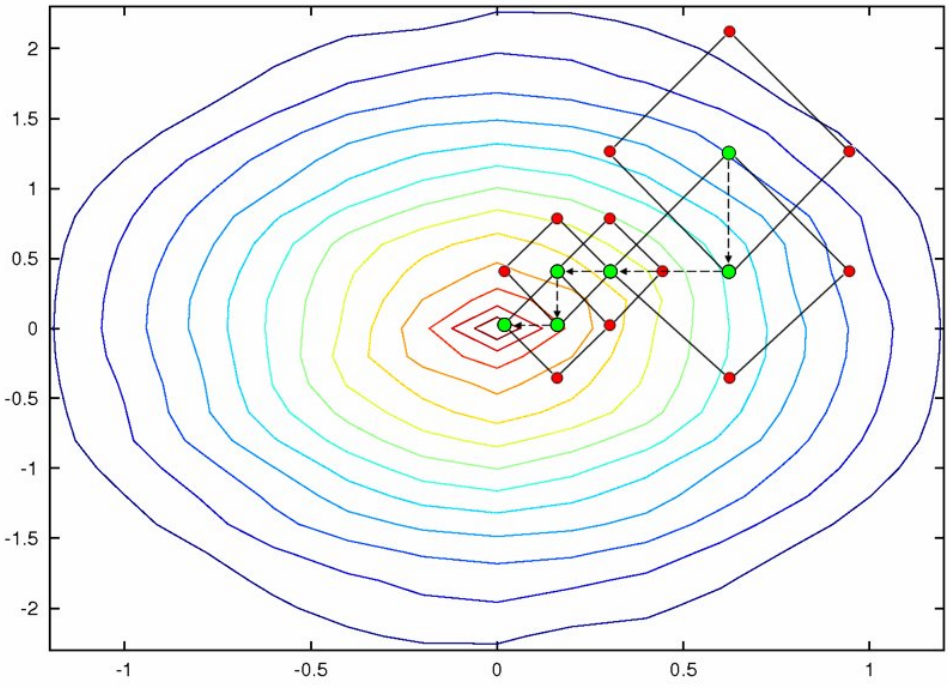

Abbildung 6: Optimierungsstrategie

4 Ähnlichkeitsmaße

4.1 Begriff der Ähnlichkeit

Der Begriff der Ähnlichkeit beschreibt nach [Sol04] allgemein den Grad der Übereinstimmung zweier Dinge in einem oder mehreren Merkmalen. Als Merkmale gelten gemeinsame Eigenschaften, die sich bei vergleichender Betrachtung in der Beschaffenheit beider Dinge erkennen lassen. Die Ähnlichkeitsbeziehung zwischen den Merkmalen ist eine graduelle und mehrdimensionale Abstufung in Qualität und Quantität der Merkmale. Daher ist es von wesentlicher Bedeutung für die Festlegung einer Ähnlichkeit, wie viele Eigenschaften zur Betrachtung herangezogen werden, wie stark diese auftreten, welches Gewicht sie bei der Beurteilung haben und wie diskret in ihrer Übereinstimmung differenziert wird. Zur Bestimmung der Ähnlichkeit wird das Verhältnis zwischen gemeinsamen und unterscheidenden Eigenschaften abgewogen. Je mehr Merkmale sich finden lassen, die den Beurteilungskriterien entsprechen, desto größer wird die *Ähnlichkeit*. Umgekehrt proportional verhält es sich mit der *Unähnlichkeit*. Bei einer vollkommenen Übereinstimmung aller erkennbaren Merkmale spricht man von ihrer Gleichheit oder der *Identität* der Dinge.

Das Ähnlichkeitsempfinden des Menschen erstreckt sich über alle Sinne der Wahrnehmung. Im Falle der optischen Ähnlichkeit bezieht sich der Begriff im Umgang mit bildhaften Darstellungen üblicherweise auf die Ähnlichkeitsrelation zwischen abgebildetem Objekt und Bild selbst oder jene zwischen zwei Bildern. Die philosophische Diskussion sieht die Schwäche des Ähnlichkeitsbegriffs in der Subjektivität der menschlichen Wahrnehmung. Es kann nicht allgemeingültig festgelegt werden, welche Eigenschaften als relevant gelten müssen, um einen sinnvollen Vergleich anzustellen. Dabei spielt vor allem eine Rolle, welche Eigenschaften sich nach persönlichem Empfinden überhaupt vergleichen lassen. Neben der Vergleichbarkeit ist auch nicht definiert, in wie vielen Eigenschaften eine Übereinstimmung gefunden werden muss, damit etwas als ähnlich gelten kann. Diese Aspekte werden beeinflusst durch das Vorwissen und Erfahrungen sowie die Intention des Betrachters. Ebenso sind erlernte Sehgewohnheiten und der kulturelle Hintergrund von Bedeutung.

Folglich muss zwischen wahrgenommener und objektiv gemessener Ähnlichkeit unterschieden werden (perceived similarity vs. judged similarity)[San99]. Während der Mensch eine kontinuierliche Wahrnehmung besitzt, muss im Bereich der messbaren Ähnlichkeit eine diskrete Einteilung der Anzahl und Art der Merkmale, der Gewichtung und Abstufung der einzelnen Merkmale, sowie die Festlegung eines Schwellwertes zwischen Ähnlichkeit und Unähnlichkeit vorgenommen werden. Dann lässt sich als

berechenbare Ähnlichkeit definierten

$$\text{Ähnlichkeit} = \sum_i \text{Gewicht}_i * \text{Merkmal}_i \quad .$$

4.2 Anwendungen

In der Signalverarbeitung wird die Ähnlichkeit zweier Signale durch das Ausmaß ihrer Differenz oder der Korrelation auf einem bestimmten Merkmal festgestellt. Im Falle der Differenz wird jedoch tatsächlich die Unähnlichkeit gemessen. Die Grundlage der Messung sind hier geometrische Abstandsmaße, die Ähnlichkeit genau dann feststellen, wenn sich die Signale durch eine Transformation aufeinander abbilden lassen und der gemessene Abstand dabei minimal wird. Die Wahl der in die Messung einfließenden Eigenschaften bestimmt dabei eindeutig einen Merkmalsraum und das auf ihn anzuwendende Maß. Solche Metriken finden sich in den Anwendungsgebieten der Sprachanalyse etwa zur Stimmmustererkennung oder bei der Arbeit mit Bildern in Computergrafik und Bildverarbeitung im Wesentlichen in den zwei folgenden Gebieten.

In der Datenbankforschung geht der Trend stark in Richtung der Multimediadatenbanken, die modalitätsübergreifendes Wissen vereinen. Es werden unter anderem auch Datenrepräsentationen in Bildform gespeichert, was neue Probleme bezüglich der Ordnung auf Bilddaten durch Sortierung und Klassifizierung sowie Schlagwort-Suchanfragen aufwirft. Klassische sprachlich-semantische Methoden genügen nicht, der menschlichen Wahrnehmung gerecht zu werden. Es muss besonders das Ähnlichkeitsempfinden des Menschen berücksichtigt werden, denn Bilder die einen inhaltlich semantischen Zusammenhang haben, müssen nicht gleichzeitig als visuell ähnlich empfunden werden.

Im Allgemeinen werden Sortierung und Suche nach den Dateieigenschaften wie Name, Datum oder Größe einer Bilddatei und durch automatische oder manuelle Bildbeschreibungen auf sprachlicher Ebene hergestellt, nicht jedoch nach ihrer visuellen Gestalt. Diese rein beschreibende Indizierung unterliegt der subjektiven Auffassung von Ähnlichkeit (vgl. [Sol04]), sowie der Mehrdeutigkeit der Sprache und ermöglicht daher keine optimale Trefferquote. Je nach angewendeten Suchbegriffen können die Ergebnisse gerade bei der Internetsuche wesentlich zu groß oder zu gering ausfallen. Bei der inhaltsbasierten Bildsuche[4] (CBIR) hingegen werden visuelle Bildmerkmale wie Farbe, Textur und Form der Inhalte eines Beispielbildes dazu herangezogen, Bilder in der Datenbank aufzufinden, die aufgrund von Gemeinsamkeiten der visuellen Eigenschaften als Ergebnis der Datenbankanfrage gelten können. In [Bar08] wird ein Verfahren vorge-

[4] Content Based Image Retrieval (CBIR)

stellt, das eine Sortierung einer Bilddatenbasis anhand der Ähnlichkeiten der Farbverläufe der Bilder durchführt.

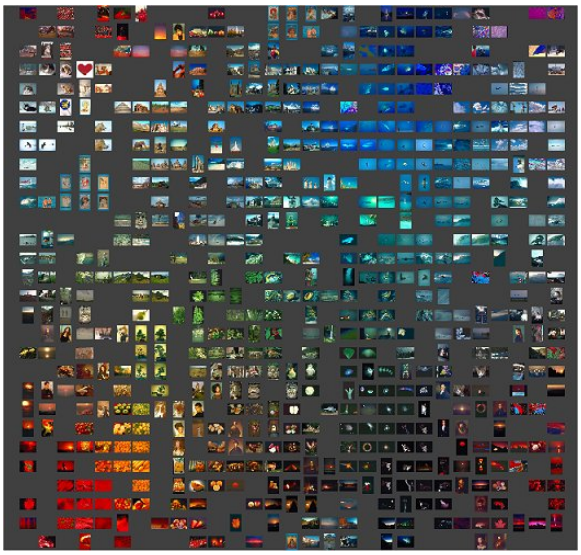

Abbildung 7: Eine mit dem Programm ImageSorter aus [Bar08] durchgeführte Sortierung von Bildern nach der Ähnlichkeit ihrer Farbverläufe

Die größte Bedeutung kommt den bildbasierten Ähnlichkeitsmaßen in der Disziplin des Rechnersehens (engl. Computer Vision) zu. Im Gegensatz zur Anwendung im Datenbankbereich ist der Bildinhalt (semantische und pragmatische Ebene) hier nicht von Bedeutung, die Ähnlichkeit wird nur auf Basis der reinen Pixelwerte (syntaktische Ebene) bestimmt [Smi06]. Ähnlichkeitsmetriken werden im Bereich von Bildverarbeitung und Rechnersehen hauptsächlich eingesetzt, um die Aufgabe der Bildregistrierung durchzuführen. Unter Bildregistrierung versteht man das in Deckung bringen von Bildern, bei welchem man nach [Bro92] vier Aufgabenbereiche unterscheidet:

Referenzielle Registrierung Sie dient der Muster- und Objekterkennung, also dem Auffinden von bekannten Strukturen im Bild. Die zu analysierenden Bilder oder Bildausschnitte werden mit Referenzbildern verglichen (Pattern Matching). Auch die Kamerakalibrierung nutzt dieses Verfahren. Dabei wird ein vermessenes Kalibrierungsmuster mit einer Bildaufnahme registriert. Aus der Transformation lassen sich die intrinsischen Kameraparameter ableiten, um eine durch das Linsensystem der Kamera verursachte Verzerrung (Distortion) auszugleichen.

Räumliche Registrierung Die Registrierung von Bildern, die aus verschiedenen Kamerasichten der gleichen Szene aufgenommen wurden, wird zur

3D-Rekonstruktion verwendet. Die Disparität zwischen zwei Stereobildern lässt auf die Tiefeninformation schließen (Struktur aus Bewegung). Beim Tracking gibt die räumliche Veränderung zwischen zwei Bildern Aufschluss über die Posebeziehung von Objekt und Kamera. Ebenso ist es möglich, bei Bewegung der Kamera einzelne Aufnahmen aus mehreren Ansichten (z.B. Panorama- oder Satellitenbilder) zu einem großen Mosaikbild zu vereinen.

Zeitliche Registrierung Durch die Registrierung von Bildern in einer zeitlichen Abfolge bei festem Kamerastandort, können Veränderungen der Szene erkannt werden. Sie findet Anwendung in der Überwachung und Analyse von dauerhaften Vorgängen. Etwa in Sicherheitssystemen um die Bewegung von Objekten im Bild zu erfassen oder in der Medizin um den Heilungsprozess zu dokumentieren.

Multimodale Registrierung Aufnahmen der gleichen Szene, die jedoch mit den unterschiedlichen Bildsensoren verschiedener Modalitäten aufgenommen wurden, werden per Registrierung in Deckung gebracht. Dieser als Fusion bezeichnete Vorgang dient dazu, durch Integration der Informationen einzelner Bilder den Gesamtinformationsgehalt zu steigern. Er findet Anwendung in der Satellitenkartografie und der Medizin, etwa bei dem Vereinen von struktureller Information der Computertomographie (CT) mit funktionaler Information der Positronen Emissionstomographie (PET).

Die Analyse durch Synthese wird nach [Zit03] nicht in die bereits genannten Bereiche eingeordnet, da hierbei die von einem Sensor erfasste Szene mit einem synthetischen Modell registriert wird. Je nach Verwendung des vorhandenen Modellwissens und der Darstellungsform können verschiedene Registrierungsarten betroffen sein.

Bei einer Bildregistrierung werden zwei Bilder durch Transformation so überlagert, dass möglichst viele korrespondierende Punkte in beiden Bildern in Übereinstimmung gebracht werden können (Homographie). Seien I_1 und I_2 Bilder mit den Merkmalen $I_1(x, y)$ und $I_2(x, y)$ an den entsprechenden Koordinaten. T ist eine Transformation von I_2 auf I_1 und d das Ähnlichkeitsmaß. Eine Übereinstimmung ist gefunden, wenn für alle verglichenen Bildpunkte gilt:

$$I_1(x, y) = I_2(T(x, y))$$
$$mit$$
$$T = \underset{T}{\operatorname{argmax}}\, d(I_1, T(I_2))$$

Dabei gilt es, die Transformation so zu optimieren, dass der durch ein angewandtes Ähnlichkeitsmaß errechnete Abstand minimal oder die Ähnlichkeit maximal wird. Dies geschieht in der Regel durch lokale Suche nach

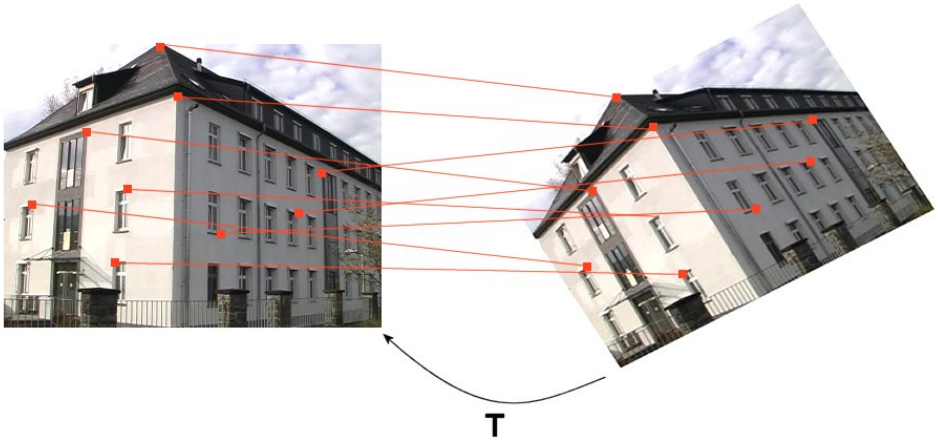

Abbildung 8: Bildregistrierung

in zwei Bildern korrespondierenden Merkmalen wie Eckpunkte oder Linien, die in einem Vorverarbeitungsschritt detektiert werden. Es ist auch möglich, mit Ähnlichkeitsmaßen eine globale Suche auf dem gesamten Bild oder einem Ausschnitt (Block-Matching) anzustreben, um eine Überlagerung zu erreichen. Darauf soll diese Arbeit aufbauen. Im Vergleich zur echten Bildregistrierung entfällt aber die Berechnung der Transformation zwischen den Bildern, da mit dem Ansatz der Analyse durch Synthese das benötigte Wissen über die Kamerapose bereits durch die Bildsynthese bekannt ist und nur noch der Ähnlichkeitswert zur Beurteilung der Übereinstimmung herangezogen wird. [Str02] unterteilt die Bildregistrierungsmethoden und damit die Ähnlichkeitsmaße in drei Klassen:

Intensitätsbasierte Verfahren Die in dieser Arbeit verwendeten intensitätsbasierten Verfahren operieren auf den Intensitäts- oder Luminanzwerten der einzelnen Pixel der zu vergleichenden Bilder. Da die Informationen über die Lage der Pixelwerte unabhängig von Inhalt und Struktur des Bildes sind, ist weder Extraktion noch Korrespondenzsuche nötig. Das ganze Bild kann global in einem Schritt ausgewertet werden und Vorverarbeitungsschritte entfallen. Die korrespondierenden Pixelwerte zweier Bilder können direkt paarweise verglichen werden, wobei Betrachtung von Differenz oder Korrelation möglich ist. Ferner gibt es den indirekt-statistischen Vergleich über die Bildhistogramme, welche die Verteilung der Pixelwerte im Bild angeben. Deren Erstellung erfordert jedoch wieder einen Vorberechnungsschritt.

Merkmalbasierte Verfahren Liegen dem Vergleich Merkmale zu Grunde, muss zunächst auf beiden Bildern eine Suche durchgeführt werden.

Typische Elemente des Merkmalsraums sind Bildpunkte, Bereiche gleicher Gradienten (Kanten), Schnittpunkte von Kanten (Ecken) oder gar Konturen und Formen, die aus einem Gebilde von Kanten bestehen können, sowie segmentierte Flächen. Dieser Vorgang kann durch manuelle Selektion oder automatisiert durch Detektionsverfahren erfolgen. Als bekannteste unter ihnen sind der Moravec-Interest-Operator, der Harris-Corner-Detektor und Scale Invariant Feature Transform (SIFT) zu nennen. Danach wird eine Zuordnung der korrespondierenden Merkmale in beiden Bildern durchgeführt und die geometrische Transformation zwischen ihnen ermittelt, um darauf die Bildüberlagerung auszuführen.

Frequenzbasierte Verfahren Die Bilder werden zunächst durch Transformation in den Frequenzraum umgewandelt. Bekanntestes Beispiel ist die Fouriertransformation, welche die Bildsignale in eine Summe ihrer Sinus- und Kosinusfequenzanteile zerlegt. Das daraus resultierende Frequenzspetrum wird dann dem Vergleich zugrunde gelegt.

Im Folgenden soll auf einige der intensitätsbasierten Ähnlichkeitsmaße eingegangen werden. Weiterführende Vergleiche und Untersuchungen finden sich in [Cox95] und [Wei03].

4.3 Abstandsmaße

Die intensitätsbasierten Abstandsmaße berechnen die Differenz zweier Bilder direkt auf den Intensitäten der einzelnen Pixel. Dazu wird aus den Differenzen der Intensitätswerte korrespondierender Pixel in beiden Bildern die Summe gebildet. Durch den absoluten Betrag der Differenz oder das Quadrat der Differenzen werden negative Werte unterdrückt. Allerdings hat die Quadrierung den Nachteil, dass sie geringe Fehler vergrößert und diese somit stärker ins Gewicht fallen. Die den Bildabstandsmaßen zugrundeliegende geometrische Ähnlichkeit ist formal wie folgt definiert [Cox95][Smi06]. Seien f,g und h Bildfunktionen, so ist d ein Distanzmaß, wenn gilt

$$\begin{aligned}
\text{Identität} \quad & d(f,g) = 0 \;\Leftrightarrow\; f = g \\
\text{Symmetrie} \quad & d(f,g) = d(g,f) \\
\text{Positivität} \quad & d(f,g) \geq 0 \\
\text{Dreiecksungleichung} \quad & d(f,g) \leq d(f,h) + d(h,g)
\end{aligned}$$

[Smi06] definiert ein Ähnlichkeitsmaß als Funktion, die zu vergleichenden Objekten eine reelle Zahl aus dem Wertebereich [0,1] zuordnet. Da es sich hierbei aber um ein Distanzmaß handelt, wird das Ergebnis maximal, wenn die Ähnlichkeit am geringsten ist. Im eigentlichen Sinne wird also die Unähnlichkeit gemessen. Um einen Ähnlichkeitswert zu erhalten,

ist das Ergebnis nach Normierung auf den Wertebereich [0,1] noch zu invertieren, damit 1 die maximale Ähnlichkeit beschreibt 0 die maximale Unähnlichkeit.

Die Differenzmaße werden aus der Minkowski-Distanzfunktion L_m abgeleitet.

$$d_{L_m}(f,g) = \left(\sum_{x,y} |(f(x,y) - g(x,y))|^m \right)^{\frac{1}{m}}$$

Für m=1 ergibt sich mit der L_1-Norm die Summe der absoluten Differenzen (SAD)[5], die auch Block- oder Manhattan-Metrik genannt wird.

$$d_{SAD}(f,g) = \sum_{x,y} |f(x,y) - g(x,y)|$$

Aus m=2 folgt die L_2-Norm, welche als die euklidische Distanzfunktion bezeichnet wird, da sie den kürzesten Abstand zwischen zwei Punkten berechnet.

$$d_{L_2}(f,g) = \sqrt{\sum_{x,y} (f(x,y) - g(x,y))^2}$$

Die quadrierte euklidische Distanzfunktion ist metrisch äqivalent zur quadratischen Distanzfunktion, auch Summe der quadratischen Differenzen (SSD)[6] genannt.

$$d_{SSD}(f,g) = \sum_{x,y} (f(x,y) - g(x,y))^2$$

Damit die Differenzmaße im gewünschten Intervall [0,1] vorliegen, sollten sie mit dem Produkt der quadratischen Pixelsummen der beiden Bilder normiert werden:

$$d_{SAD_{normalized}}(f,g) = \frac{d_{SAD}}{\sqrt{\sum_{x,y} f(x,y)^2} \sqrt{\sum_{x,y} g(x,y)^2}}$$

$$d_{SSD_{normalized}}(f,g) = \frac{d_{SSD}}{\sqrt{\sum_{x,y} f(x,y)^2} \sqrt{\sum_{x,y} g(x,y)^2}}$$

[5] Sum of Absolute Differences (SAD)

[6] Sum of Squared Differences (SSD) oder Squared Error (SE)

4.4 Korrelationsmaße

Mit den Korrelationsmaßen [Wei03] wird festgestellt, ob ein linearer Zusammenhang zwischen zwei Werten besteht. Dazu wird die Summe über die Produkte der Pixelintensitäten beider Bilder erstellt. Die Normierung führt auch hier, wie bei den Abstandsmaßen, zur Skalierung des Intervalls. Allerdings liegt der Wertebereich der Korrelation bei [-1,1], denn es kann sowohl der positive Zusammenhang zweier Werte gemessen werden, als auch der negative. Für einen Wert von 1 liegt ein linearer Zusammenhang vor, die Bilder sind maximal ähnlich. Für einen Wert von 0 existiert kein linearer Zusammenhang, die Bilder sind maximal unähnlich. Der Fall des negativen linearen Zusammenhangs tritt beim Vergleich eines Bildes mit seinem Negativbild auf. Da dies jedoch für das Tracking irrelevant ist, können negative Werte durch den Betrag der Summe korrigiert werden.

Die Korrelation zweier Werte wird bestimmt durch den Korrelationskoeffizienten[7], welcher sich aus der normierten Kovarianz errechnet

$$Kor(f,g) = \frac{Cov(f,g)}{\sqrt{Var(f)}\sqrt{Var(g)}}$$

wobei gilt

$$\begin{aligned} \text{Identität} \quad & Kor(f,g) = 1 \;\Leftrightarrow\; f = g \\ \text{Symmetrie} \quad & Kor(f,g) = Kor(g,f) \\ \text{Negativität} \quad & Kor(f,-f) = -1 \quad . \end{aligned}$$

Die normierte Kreuzkorrelation (CC)[8]

$$d_{CC}(f,g) = \frac{\sum_{x,y} f(x,y) * g(x,y)}{\sqrt{\sum_{x,y} f(x,y)^2}\sqrt{\sum_{x,y} g(x,y)^2}}$$

lässt sich mit den Mittelwerten f_μ und g_μ der beiden Bilder f und g zur normalisierten Kreuzkorrelation (NCC)[9] erweitern

$$d_{NCC}(f,g) = \frac{\sum_{x,y}(f(x,y) - f_\mu) * (g(x,y) - g_\mu)}{\sqrt{\sum_{x,y}(f(x,y) - f_\mu)^2}\sqrt{\sum_{x,y}(g(x,y) - g_\mu)^2}} \quad .$$

Sie ist die mittelwertfreie Variante und daher unempfindlich gegenüber Helligkeits- und Kontrastschwankungen. Da für jede Differenzbildung jedoch der Mittelwert bekannt sein muss, ist eine vorherige Berechung nötig und die Summe über alle Pixelwerte muss zweimal durchlaufen werden.

[7] http://de.wikipedia.org/wiki/Korrelationskoeffizient
[8] Cross Correlation (CC)
[9] Normalized Cross Correlation (NCC)

Der Verschiebungssatz der Statistik[10] schafft hier Abhilfe um eine Vorberechnung zu vermeiden und den Rechenaufwand zu senken, indem der Mittelwert aus der Summe entfernt wird. Mit n als Anzahl der Pixel im Bild gilt für die Kovarianz

$$Cov(f,g) = \sum_{x,y}(f(x,y) - f_\mu)(g(x,y) - g_\mu) = \sum_{x,y}(f(x,y)g(x,y)) - nf_\mu g_\mu$$

und die Varianz

$$Var(f) = \sum_{x,y}(f(x,y) - f_\mu)^2 = \sum_{x,y} f(x,y)^2 - nf_\mu^2$$

und daher lässt sich die NCC umformen zu

$$d_{NCC}(f,g) = \frac{\sum_{x,y}(f(x,y)g(x,y)) - nf_\mu g_\mu}{\sqrt{\sum_{x,y} f(x,y)^2 - nf_\mu^2}\sqrt{\sum_{x,y} g(x,y)^2 - ng_\mu^2}} \quad .$$

4.5 Farbabstand

Die bisher betrachteten Ähnlichkeitsmaße wurden nur auf die Grauwertintensitäten angewendet. Der Abstand zweier Farben im hier betrachteten RGB-Farbraum ist der Abstand ihrer Farbvektoren, genauer gesagt, die Länge ihres Differenzvektors. Zur Berechnung wird die euklidische Distanz im 3D-Raum zugrunde gelegt. Für jedes Pixel im Bild existiert ein Farbvektor

$$f(x,y) = \begin{pmatrix} f_R(x,y) \\ f_G(x,y) \\ f_B(x,y) \end{pmatrix}$$

sodass die Differenz der drei Farbwerte pro Pixel bestimmt werden kann als Differenzvektor

$$\Delta_{RGB}(x,y) = \begin{pmatrix} \Delta_R(x,y) \\ \Delta_G(x,y) \\ \Delta_B(x,y) \end{pmatrix} = \begin{pmatrix} f_R(x,y) - g_R(x,y) \\ f_G(x,y) - g_G(x,y) \\ f_B(x,y) - g_B(x,y) \end{pmatrix}$$

Der Farbabstand (CD)[11] ist somit definiert als die Summe der Beträge aller Farbdifferenzen

$$d_{CD}(f,g) = \frac{\sum_{x,y}\sqrt{\Delta_R(x,y)^2 + \Delta_G(x,y)^2 + \Delta_B(x,y)^2}}{n}$$

normiert durch die Anzahl n der Farbpixel im Bild.

[10] http://de.wikipedia.org/wiki/Verschiebungssatz_(Statistik)
[11] Color Difference (CD)

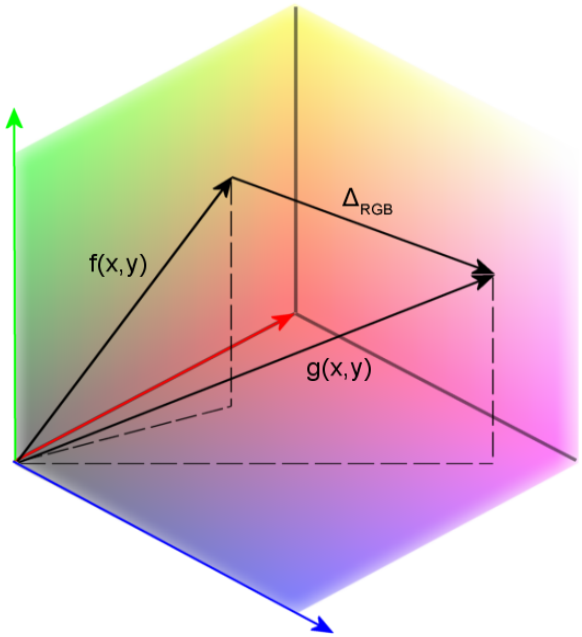

Abbildung 9: Abstand der Farbvektoren

4.6 Histogramme

Histogramme geben Aufschluss über die Häufigkeit des Vorkommens von Farbwerten im Bild (bei Grauwertbildern analog über die Grauwerte). Für jeden im Bild vorkommenden Farbwert wird eine Summe über die Anzahl seines Auftretens gebildet und im Histogramm in sogenannten Urnen (engl. bins) gespeichert. Ein solches Histogramm mit absoluter Häufigkeitsangabe wird durch Normierung mit der Gesamtpixelzahl in eines mit relativen Häufigkeiten umgewandelt, sodass die Summe aller Häufigkeiten eins ergibt. Um diese Histogramme auf Ähnlichkeit zu vergleichen, lassen sich die aus der Minowski-Distanz abgeleiteten Abstandsmaße ebenso anwenden, wie die Korrelation. Dabei wird statt der Pixelintensitäten die Häufigkeit eines jeden Farbwertes beider Histogramme paarweise miteinander verglichen.

Sind F und G Histogramme der Bilder f und g, die Anzahl der Farbwerte im Bild n und halten F_n und G_n die Häufigkeit genau eines Farbwertes des jeweiligen Bildes, dann kann die Summe der quadratischen Differenzen (SSD) wie beschrieben angewendet werden:

$$d_{SSD}(F,G) = \sum_n (F_n - G_n)^2 \quad .$$

Als Variante der L_1-Norm des geometrischen Abstands gilt der sogenannte Histogrammschnitt (HI)[12]

$$d_{HI}(F,G) = \sum_n \min\{F_n, G_n\}$$

der die maximale Übereinstimmung durch den Ergebniswert 1 anzeigt. Auf einem Histogramm mit relativer Häufigkeit berechnet, braucht dieses Maß nicht mehr normiert zu werden, da sich die Werte des Histogramms bereits im Intervall [0,1] befanden und die Summe der relativen Häufigkeiten nicht größer als 1 werden kann. Neben den geometrischen Distanzen können auch Verfahren aus der Statistik eingesetzt werden. Ein bekanntes Verfahren für Histogrammvergleiche ist der Chi-Square-Test (χ^2)

$$d_{\chi^2}(F,G) = \sum_n \frac{(F_n - G_n)^2}{G_n}$$

von dem mehrere Versionen in der Literatur existieren. Die Bildverarbeitung kennt ein breites Spektrum an weiteren Möglichkeiten für verschiedenartige Histogrammvergleiche, das sich besonders aus dem Bereich der Objekterkennung heraus entwickelt hat. Einen tiefgreifenderen Überblick geben [Pau01] und [Rub01].

[12] Histogram Intersection (HI)

5 Stilisierung

Neben dem Finden eines für das bildbasierte markerlose Tracking geeigneten Ähnlichkeitsmaßes, ist es ein weiterer Untersuchungsansatz dieser Arbeit, ob das Ergebnis der Ähnlichkeitsberechnung durch eine Stilisierung der Bilder vor dem Vergleich verbessert werden kann. Es steht vor allem die Frage im Raum, in wieweit die Präzision des Trackings dadurch zu erhöhen ist. Die Stilisierung soll als Angleichung von Kamerabild und dem gerenderten Bild aus der Synthese dienen. Dies geschieht je nach Darstellungsform durch Hinzufügen oder Reduktion von Information im Bild. Der Fokus liegt dabei auf nicht photorealistischen Stilisierungsmethoden.

Unter nicht photorealistischem Rendern (NPR)[13] versteht man die Stilisierung von Bildern durch Veränderungen in Form, Farbe, Struktur, Schattierung und Licht, sodass es zu einer inhaltlichen Abstraktion oder Hervorhebung von Details kommt. NPR steht dabei jedoch nicht im Gegensatz zur photorealistischen Computergraphik [Swe02].

Die Techniken des nicht photorealistischen Renderns wurden unter anderem entwickelt, um die Verständlichkeit von Bildern zu steigern. Photorealistische Darstellungen sind oft zu detailreich um schnell und zuverlässig Informationen zu vermitteln. Subjektive und kulturelle Wahrnehmung führen zu Interpretationsproblemen, weshalb man mit NPR versucht, die Darstellung für den jeweiligen Anwendungszweck eines Bildes zu optimieren. Die Wahrnehmung wird zu diesem Zwecke durch Strukturierung und Abstraktion der Bildinformationen gezielt gelenkt, indem Details reduziert oder betont werden. Die Grundelemente sind dabei flächige Farben und Linien. Die häufigsten Anwendungen finden sich im Bereich der technischen Illustrationen, piktogrammgestützter Texte wie Anleitungen und der Darstellung medizinischer Daten.

Der zweite Schwerpunkt von NPR liegt auf den künstlerischen Anwendungen. In diesem Bereich wird angestrebt, gestalterische Werkzeuge zu simulieren, die es erlauben ein Bild so aussehen zu lassen als wäre es handproduziert. Die Palette reicht über zahlreiche Kunsttechniken und Stile wie Pinsel, Wasserfarben, Bleistift und Tusche bis zur Nachahmung verschiedener Trägermedien wie Papier und Leinwand. Neben der automatischen Generierung solcher Bilder existieren auch Systeme, bei denen der Benutzer interaktiv Bilder erstellen kann. Nicht zuletzt ist NPR in der Form des Cartoon-Shadings in Filmen bekannt geworden, die sich durch typische Farbabstufungen und Silhouettendarstellung auszeichnen.

Nicht photorealistisches Rendern optimiert Bilder für die menschliche Wahrnehmung. Ob dieser Nutzen auch beim informationstechnischen Ähnlichkeitsvergleich von Bildern wirkungsvoll ist, muss die Untersuchung

[13] Non-Photorealistic Rendering (NPR)

zeigen. Im Folgenden werden die fünf Darstellungsformen vorgestellt, die in dieser Arbeit zum Einsatz kommen. Abbildung 12 zeigt jeweils das Kamerabild und das Rendering einer Darstellungsform im Vergleich. Dabei werden künstlerische Verfahren als ungeeignet ausgeschlossen, da sie gerade nicht der Vereinheitlichung dienen, sondern durch zufallsgenerierte Strukturen wie sie etwa bei Pinselstrichen oder Schraffuren entstehen, einen einzigartigen Charakter des Bildes erzeugen sollen.

5.1 Diffuses Shading

Als Vergleichsreferenz für die anderen Stilisierungsformen soll eine möglichst realitätsnahe Darstellung dienen, die mit dem unstilisierten Originalbild der Kamera direkt verglichen werden kann. Die Oberfläche des zu trackenden Objekts wird als ideal diffus angenommen, das heißt einfallendes Licht wird in alle Richtungen mit der gleichen Intensität gestreut. Es wird folglich für das Rendering des synthetischen Bildes eine einfache diffuse Beleuchtung nach dem Lambert-Modell angewendet, da der Realitätsgrad dieses Shadings der Beschaffenheit des physischen Objekts am nächsten kommt. Diese Art der Beleuchtung ist lediglich abhängig vom Cosinus des Winkels zwischen den Oberflächen- oder Eckpunktnormalen des Modells und dem einfallenden Lichtstrahl, jedoch nicht vom Beobachtungswinkel. Der Cosinus des Winkels zwischen zwei Vektoren lässt sich als Skalarprodukt darstellen. Es gilt also, dass die Lichtintensität `I` an einem Punkt der Oberfläche durch das Skalarprodukt von Lichtvektor $\vec{l}$ und Normale $\vec{n}$ des zu beleuchtenden Punktes berechnet wird

$$I = \frac{\vec{l} \cdot \vec{n}}{|\vec{l}||\vec{n}|} \quad .$$

Je flacher das Licht auf den Punkt trifft, desto dunkler wird er erscheinen. Senkrechter Lichteinfall erzeugt die größte Helligkeit. Werden statt der gesamten Oberfläche eines Polygons (Flat-Shading) die Eckpunkte beleuchtet (Gouraud-Shading), wird die Intensität der Farbwerte an den Eckpunkten noch über die Fläche interpoliert.

5.2 Selbstverschattung

Schatten spielen eine wesentliche Rolle in der menschlichen Wahrnehmung bei der Einschätzung von Entfernungen und dem Erkennen der Lage von Objekten. Das Ergänzen des diffusen Shadings durch Selbstverschattung steht ebenfalls im Gegensatz zu den folgenden informationsreduzierenden Verfahren. Damit soll getestet werden, in wieweit zusätzliche Informationen über die geometrische Objektbeschaffenheit und die Lichtverteilung auf dem Objekt für die Bestimmung der Ähnlichkeit von Nutzen sein können.

Zur Erzeugung der Schatten wird auf das Verfahren des von [Wil78] vorgestellten *Shadow-Mapping* zurückgegriffen. Da es im Kern nur auf der Erstellung von Texturen aus dem Tiefenbuffer beruht, ist es schnell und ohne großen Aufwand zu realisieren. Es wird von OpenGL unterstützt[14] und läuft vollständig auf der Hardware. Für Echtzeitrendering ist es besonders geeignet, da es unabhängig von der Komplexität der Geometrie der Szene arbeitet. Als Einschränkung gilt, dass nur Spotlichtquellen verwendet werden können, was aber kein Problem darstellt, da dies der verwendeten realen Lichtquelle entspricht. Auch ist das Verfahren von der Texturauflösung abhängig, so dass es bei zu kleiner Shadow Map zu Aliasing-Effekten an den Schattenrändern kommt. Es gibt zahlreiche Erweiterungen und Verbesserungen des Standard-Shadow-Mapping Algorithmus, doch für die vorliegende Arbeit soll die einfache Form genügen.

Die Idee baut auf der Tatsache auf, dass Objektpunkte die im Schatten liegen, von der Lichtquelle aus gesehen nicht sichtbar sind. Zuerst wird die virtuelle Kamera an die Position der Lichtquelle gesetzt und die Szene aus Sicht der Lichtquelle gerendert. Dabei sind nur die Tiefenwerte der Szene von Interesse. Sie werden aus dem Tiefenbuffer ausgelesen und in einer Textur (Shadow Map) gespeichert. Die Shadow Map muss nur einmal generiert werden, wenn sich das Verhältnis von Objekt und Lichtquelle nicht ändert und ist daher unabhängig vom Betrachter. Danach kann die Szene aus der eigentlichen Kamerasicht gerendert werden. Die Entfernung jedes Objektpunktes zur Lichtquelle wird nun mit dem korrespondierenden Tiefenwert in der Shadow Map verglichen. Dazu ist es erforderlich, die Pixelkoordinaten der Kamera wie folgt in das Koordinatensystem der Lichtquelle zu transformieren.

$$\begin{pmatrix} s \\ t \\ r \\ q \end{pmatrix} = \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \text{Projektions-} \\ \text{matrix der} \\ \text{Lichtquelle} \end{pmatrix} \begin{pmatrix} \text{Eyeview-} \\ \text{matrix der} \\ \text{Lichtquelle} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Dabei werden Texturkoordinaten im Intervall [0,1] generiert, die den Zugriff auf die Shadow Map ermöglichen. Durch die Transformation entsprechen die Pixelkoordinaten s/q und t/q nun denjenigen, an deren Stelle in der Shadow Map der zur transformierten Tiefe r/q gehörende Tiefenwert aus Sicht der Lichtquelle zu finden ist. Ist die Entfernung gleich groß, kann er von der Lichtquelle aus gesehen werden und wird folglich beleuchtet. Ist die Entfernung in der Shadow Map jedoch geringer als die tatsächliche Entfernung zur Lichtquelle, dann wird der Punkt von einem Objekt verdeckt und liegt im Schatten. Durch Ungenauigkeiten in den Tiefenwerten kann es bei diesem Vorgehen zu Artefakten und flackernden Schatten kom-

[14] Shadow Extension (http://www.opengl.org/registry/specs/ARB/shadow.txt)

men. Daher empfiehlt es sich, vor dem Vergleich einen Offset-Wert auf die Tiefenwerte der Shadow Map zu addieren.

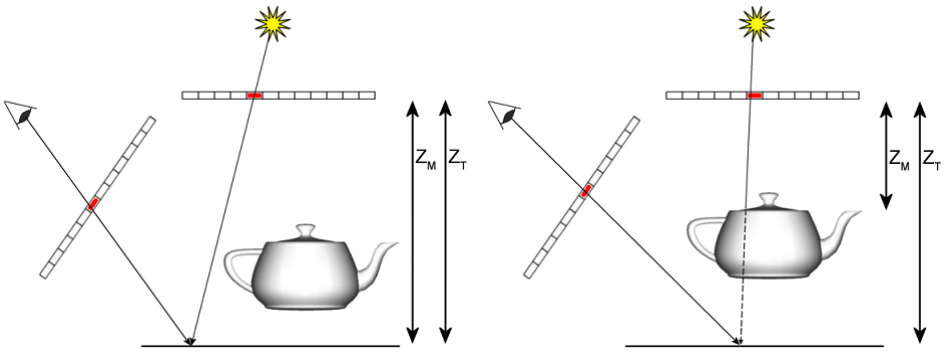

Abbildung 10: Shadow Mapping mit gespeichertem Tiefenwert in Shadow Map Z_M und transformierter Tiefe Z_T, links: kein Schatten da $Z_M = Z_T$, rechts: Schatten da $Z_M < Z_T$

5.3 Toon Shading

Eine informationsreduzierende Darstellungsvariante soll die Untersuchung ermöglichen, ob eine Abstraktion der Farb- oder Graustufenverteilung im Bild einen Vorteil bei der Ähnlichkeitsberechnung bringt. Dazu wird eine Reduzierung der Farbabstufungen des Objekts angestrebt, wobei jegliche kontinuierliche Farbverläufe eliminiert und nur flächige Farbbereiche mit harten Übergängen dargestellt werden. Dies leistet das Toon Shading [Fer06], bei dem üblicherweise eine Farbe ausgewählt wird, die je nach Lichtintensität auf der Objektoberfläche in wenigen Tönen zwischen Schwarz und Weiß auf das Objekt geshaded wird. Die Auswahl des Farbtons kann auch mit Hilfe einer 1D-Textur (Color-Ramp-Map) bestimmt werden, die alle Farbabstufungen enthält, welche je nach errechnetem Intensitätswert ausgelesen werden. Dieses Verfahren wird auch Cel-Shading genannt, da es den Eindruck eines zweidimensionalen Comics vermittelt.

In der konkreten Umsetzung werden Shader dazu verwendet eine entsprechende per Fragment Beleuchung durchzuführen. Die Entscheidung über die anzuwendende Farbstufe wird durch die Lichtintensität am zu beleuchtenden Oberflächenpunkt getroffen. Auch hier liegt das diffuse Beleuchtungsmodell der Berechnung zu Grunde. Der Vertexshader liest dazu die Position der Standardlichtquelle von OpenGL aus und berechnet den Lichtvektor vom transformierten Eckpunkt zur Lichtquelle. Zusätzlich stellt er die transformierte Eckpunktnormale bereit. Lichtvektor und Normale werden für das aktuelle Fragment interpoliert und an den Fragmentshader weitergegeben. Dieser bestimmt nach der Normierung beider Werte ihr Skalarprodukt und fängt negative Ergebnisse ab. Danach wird den In-

tensitäten ein Farbton zugewiesen. Ein Skalarprodukt nahe 1 zeigt eine hohe Intensität an. Solch ein Punkt wird mit einer Farbe nahe Weiß geshaded, um den Eindruck eines spekularen Highlights zu erzeugen. Dies ist jedoch nicht relevant, da beim Tracking ein diffuses physisches Objekt verwendet wird. Intensitäten nahe 0 werden mit sehr dunklen Tönen gezeichnet, um Schatten zu simulieren. Für Intensitätswerte dazwischen können je nach gewünschter Anzahl an Farbabstufungen beliebige Schwellwerte gesetzt werden. Für die Stilisierung auf der Seite des Kamerabildes wird analog vorgegangen, nur dass hier bei der Auswahl der Farbstufe der Wert des Skalarproduktes durch die Grauwertintensität des jeweiligen Pixels ersetzt wird, das geshaded werden soll.

5.4 Gooch Shading

Eine weitere Darstellungsform soll zur Beseitigung von Schatten und damit zur Aufhellung dunkler Bereiche am Objekt führen. Zu diesem Zweck wurde das Gooch Shading [Goo98] entwickelt, um bei technischen Illustrationen in gering oder nicht beleuchteten Bereichen die Erkennbarkeit zu erhöhen. Vollkommen schwarze oder weiße Bereiche kommen beim Shading des Objekts nicht vor. Stattdessen werden eine warme und eine kalte Farbe definiert, um Form und Krümmungsverlauf der Oberfläche zu kennzeichnen. Sie werden gewichtet auf die Grundfarbe des Objekts gerechnet. Flächen mit kaltem Farbton stehen dabei für vom Licht abgewandte Bereiche, diejenigen mit warmem Farbton für dem Licht zugewandte Bereiche. Dieses Beleuchtungsmodell wird daher auch Cool-to-Warm Shading genannt. Die so gewonnenen Kalt- und Warmschattierungen werden dann wiederum mit Hilfe des diffusen Reflektionsterms über das Objekt interpoliert. Die Gooch-Beleuchtungsformel lautet

$$k_{cool} = k_{blue} + \alpha * k_{diffuse}$$

$$k_{warm} = k_{yellow} + \beta * k_{diffuse}$$

$$k_{final} = \left(\frac{1 + N \bullet L}{2}\right) * k_{cool} + \left(1 - \frac{1 + N \bullet L}{2}\right) * k_{warm}$$

mit

k_{blue} - kalter Farbton
k_{yellow} - warmer Farbton
$k_{diffuse}$ - Grundfarbe des Objekts
α - Gewicht mit dem Grundfarbe und Kaltton verrechnet werden
β - Gewicht mit dem Grundfarbe und Warmton verrechnet werden.

Nach dieser Berechnung kann ein einzelnes Highlight hinzugefügt werden. Dessen Intensität lässt sich gemäß spekularem Term des Phong-Beleuchtungsmodells aus Skalarprodukt von reflektiertem Lichtstrahl $\vec{r}$ und

Sichtvektor $\vec{v}$ sowie der Glanzzahl n berechnen als $I = (\vec{r} \cdot \vec{v})^n$. Je höher die Übereinstimmung von reflektiertem Lichtstrahl und Sichtvektor, desto größer ist der Effekt des Highlights. Die Glanzzahl n steuert dabei die Ausdehnung des Highlights um den reflektierten Lichtstrahl. Mit zunehmender Glanzzahl nimmt die Stärke ab.

Die Umsetzung erfolgt auch hier als Shader. Der Vertexshader hat die Aufgabe, nach der Transformation der Vertices und Normalen das positive Skalarprodukt zwischen Lichtvektor und Eckpunktnormale zu berechnen. Falls ein Highlight erwünscht ist, werden Sicht- und Reflektionsvektor hergeleitet, die für die Erzeugung des Highlights im Fragmentshader nötig sind.

Der Fragmentshader übernimmt diese Werte vom Vertexshader, gewichtet die Farben mit der Grundfarbe des Objekts und interpoliert sie anhand des Skalarprodukts zwischen Normale und Lichtvektor zur endgültigen Ausgabefarbe. Auf diese wird noch der Wert des Highlights aus Skalarprodukt zwischen Reflektions- und Sichtvektor addiert. Für die Stilisierung auf der Seite des Kamerabildes wird auch hier das Skalarprodukt durch die Grauwertintensität der Bildpixel ersetzt.

5.5 Kantenbilder

Die stärkste Vereinfachung der Bildinformation soll durch Kantenbilder erreicht werden. Durch das Reduzieren auf die reinen Konturen des Objekts soll sich zeigen, ob bildbasiertes Tracking auch dann noch möglich ist, wenn keinerlei Informationen über die Objektoberfläche vorliegen. Bei Kanten handelt es sich um Diskontinuitäten im Verlauf von Grau- oder Farbwerten im Bild. Kanten sind also dort im Bild zu finden, wo eine ausreichend große Änderung der benachbarten Pixelwerte vorliegt. Diese Änderungen (Gradienten) können über Differenzbildung in x- und y-Richtung für jedes Pixel ermittelt werden. Die Größe der Ableitungen, der Gradientenbetrag, wird mit einem vordefinierten Schwellwert verglichen. Ist die Differenz ausreichend groß, wird an dieser Stelle ein Kantenpixel gesetzt.

$$Gradient_x = Pixel(x, y) - Pixel(x - 1, y)$$

$$Gradient_y = Pixel(x, y) - Pixel(x, y - 1)$$

$$Gradientenbetrag = |Gradient_x| + |Gradient_y|$$

Der Sobeloperator ist ein Kantendetektor mit Ableitung erster Ordnung. Er basiert auf einer 3x3 Filtermaske, welche mit dem Ausgangsbild gefaltet wird (Konvolution). Eine binominale Glättung zur Rauschminderung des Bildes ist bereits enthalten, was gerade bei Kameraaufnahmen von Vorteil

ist. Seine Richtungsabhängigkeit macht es nötig, zwei getrennte Ableitungen in horizontaler und vertikaler Richtung durchzuführen und zu kombinieren.

$$Sobel_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad Sobel_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Eine deutlichere Darstellung der Kanten durch Verbreiterung lässt sich realisieren, indem das Bild vor der Kantendetektion mit einem einfachen Mittelwertfilter bearbeitet wird. Auf Verfahren, die zur Silhouettenerkennung im Objektraum auf der Geometrie oder auf den Tiefen- und Normalenbuffer arbeiten, wurde verzichtet. Diese würden andere und vollständigere Kantenzüge erkennen als die Kantendetektoren auf dem Kamerabild, was der angestrebten Vereinheitlichung nicht zugute kommen würde.

Die Kanten werden beim Vergleich als reine Pixelinformation betrachtet und nicht wie bei merkmalsbasiertem Tracking in parametrisierter Form. Dennoch ist es möglich bei der Kantenerkennung einen weiteren Weg zu gehen, denn mit

$$Gradientenrichtung = \arctan(\frac{Gradient_y}{Gradient_x})$$

kann der Winkel der Kantenrichtung festgestellt werden und dessen Häufigkeit in einem Winkelhistogramm über das Bild festgehalten werden. So ist auf Basis des Kantenbildes ebenfalls ein Histogrammvergleich möglich. Dazu wird der Winkelkreis $[0°, 360°]$ in Segmente unterteilt und den Kantenrichtungen zur grafischen Darstellung die Farbe des Segments zugeordnet, in dem sie liegen.

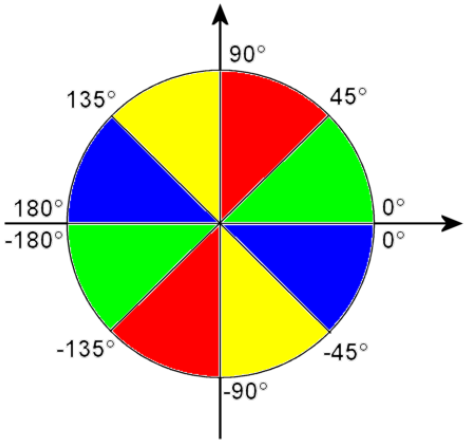

Abbildung 11: Farbliche Kodierung der Kantenrichtung in 8 Winkelsegmenten

Abbildung 12: Stilisierung, Kamerabild (links) und gerendertes Bild (rechts). Von oben nach unten: Diffuses Shading mit Selbstverschattung, Toon Shading, Gooch Shading, Kantenbilder.

6 Umsetzung

6.1 Entwurf

In diesem Kapitel soll die programmtechnische Umsetzung des markerlosen Trackings vorgestellt werden. Im ersten Schritt gilt es, die zur Realisierung benötigten Hauptkomponenten zu identifizieren und ein Modell der internen Kontroll- und Datenflüsse zu erstellen. Das zu erstellende System soll folgende grundlegende Funktionen umfassen:

- Zugriff auf Kamerabilder ermöglichen
- Rendern von Objekten mit und ohne Shader
- Stilisierung der Videoframes mit Shadern
- Berechnung der Ähnlichkeit zwischen zwei Bildern
- Suche und Optimierung der Pose
- Überlagerung des Kamerabildes mit dem Objekt
- Gleichzeitiges Darstellen von stilisiertem Kamerabild und Objekt

In Abbildung 13 ist das dazu entworfene Aktivitätsdiagramm zu sehen. Die Aktionen *Videoframe auslesen*, *Videoframe mit Objekt überlagern* und *Objekt rendern* benötigen zur Ausführung alle Zugriff auf die Renderpipeline und werden daher in OpenGL Umgebungen ausgeführt. Dabei ist das Rendern des synthetischen Bildes von der Darstellung des Kamerabildes mit Überlagerung und der Darstellung des stilisierten Kamerabildes zu trennen, damit die Bedingung der gleichzeitigen Ausgabe auf dem Bildschirm erfüllt werden kann. Es müssen folglich drei Rendering-Umgebungen erstellt werden, die wie in Abschnitt 6.3 beschrieben miteinander interagieren. Diejenige Umgebung, welche das Kamerabild mit Überlagerung ausgeben soll, erhält durch Verwendung des ARToolKit (Abschnitt 6.2) Zugang zur Videoquelle.

Bei der Aufgabe der Stilisierung sollen die Ergebnisse von *Videoframe auslesen* und *Objekt rendern* simultan unter Verwendung von Shadern stilisiert werden. Die Aktion *Stilisierung* muss daher in der Umgebung für das gerenderte Objekt und auch in der Umgebung, die das stilisierte Kamerabild anzeigt, verfügbar sein. Dazu ist es sinnvoll, eine Shader-Klasse für das Laden und Anwenden der Shader zu erstellen, die in beiden Kontexten verwendet werden kann (Abschnitt 6.4). Die verbleibenden Aktionen *Ähnlichkeit berechnen* (Abschnitt 6.5) und *Pose generieren* (Abschnitt 6.6) werden unabhängig von OpenGL als reine CPU Implementierung umgesetzt. Da diese beiden Schritte für jedes synthetische Bild ausgeführt werden, bietet es sich jedoch an, sie in der Displayschleife der OpenGL Umgebung von *Objekt rendern* aufzurufen, wo auch das synthetische Bild erstellt wird.

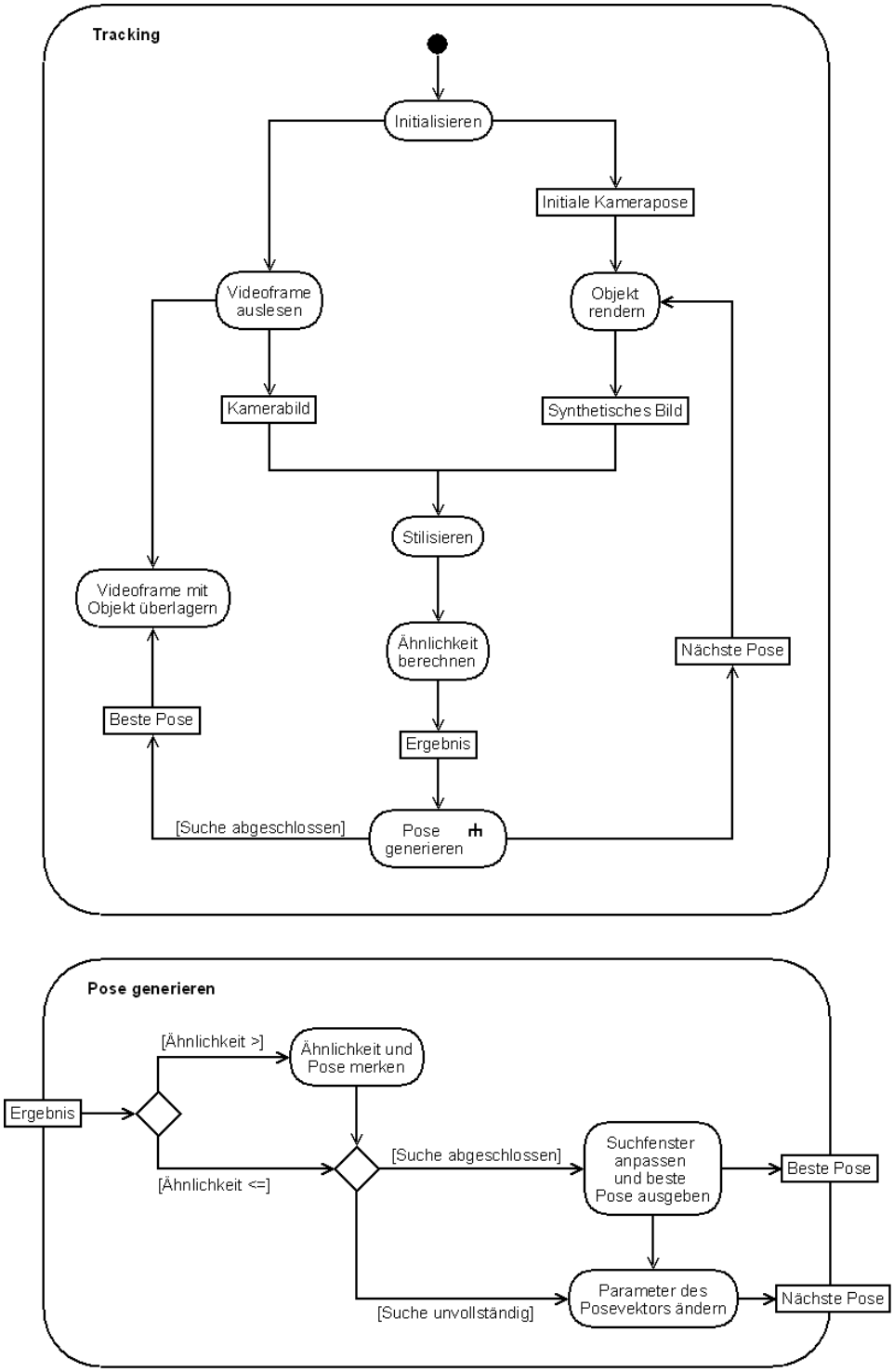

Abbildung 13: Aktivitäten

6.2 ARToolKit

Die Einbindung der Videoquelle soll das ARToolKit [ArLib] übernehmen. Es handelt sich dabei um eine Software-Bibliothek in C/C++ die als Grundlage zur Erstellung von Augmented Reality Anwendungen mit markerbasiertem Tracking dient. Zunächst ist die Videoquelle zu initialisieren. Mit `arVideoOpen` wird der Pfad zu der Bildquelle geöffnet, die wahlweise eine Kamera oder eine Videosequenz sein kann. Der Befehl `arParamLoad` lädt die benötigten intrinsischen Kameraparameter aus einer externen Datei, welche dann von `arInitCparam` initialisiert werden. Der Aufruf der Funktion `arVideoCapStart` startet den Ausleseprozess der Videoframes und `arglCameraFrustumRH` erstellt aus den eingelesenen Kameraparametern eine Projektionsmatrix für OpenGL.

Damit ist die Initialisierung abgeschlossen. Die folgenden Anweisungen werden in einer OpenGL Displayschleife verarbeitet. Mit der Funktion `arVideoGetImage` wird ein einzelnes Frame der Kamera zwischengespeichert. Dieses kann zur Darstellung mit `arglDispImage` in die OpenGL Pipeline ausgegeben werden. An dieser Stelle kann von OpenGL ein Objekt gerendert und *Videoframe mit Objekt überlagern* ausgeführt werden. Der Befehl `arVideoCapNext` fordert das nächste Frame an.

6.3 Qt

Die plattformübergreifende Klassenbibliothek Qt [QtLib] ist eine umfangreiche Sammlung von Klassen zur Programmierung von Benutzeroberflächen und beinhaltet viele weitere Funktionen wie Datenbank- oder Netzwerkunterstützung. Sie wird in dieser Arbeit eingesetzt, um die Anforderung zu erfüllen, die drei Einheiten

- Mit Objekt überlagertes Kamerabild
- Stilisiertes Kamerabild
- Stilisiertes Objekt

gleichzeitig darstellen zu können und zusätzlich benötigte Steuerelemente der Benutzerschnittstelle (GUI)[15] unterzubringen. Qt ermöglicht diese Umsetzung durch die Bereitstellung zweier wesentlicher Konzepte.

OpenGL Kontexte Mit der im OpenGL Modul von Qt enthaltenen Klasse `QGLWidget` ist es möglich, mehrere OpenGL Kontexte in einem Fenster laufen zu lassen, die gleichzeitig Inhalte rendern können. Übergibt man im Konstruktor von `QGLWidget` einen Zeiger auf andere dieser Widgets, so sind sie in der Lage, untereinander auf gemeinsame OpenGL-Daten wie

[15] engl. Graphical User Interface (GUI)

Texturobjekte und Displaylisten zuzugreifen. Daher werden also ein *Video-Widget* für das Kamerabild mit überlagertem Objekt, ein *Filter-Widget* für das stilisierte Kamerabild und ein *Objekt-Widget* für die Darstellung der gerade gerenderten und stilisierten Pose des Objekts erzeugt. Eine im Video-Widget durch den OpenGL Aufruf `glCopyTexImage2D` oder durch Aufruf von `glCopyTexSubImage2D` erstellte Textur mit dem aktuell angezeigten Kamerabild, kann im Filter-Widget durch Zugriff auf das entsprechende Texturobjekt wieder ausgelesen werden. Dort wird die Textur einem Shader zur Stilisierung übergeben und das stilisierte Kamerabild angezeigt.

Im Objekt-Widget kann analog zum ersten Schritt auf das dann stilisierte Bild zugegriffen werden. Das Objekt-Widget hat die Aufgabe, das synthetische Bild zu rendern und des Weiteren, es mit dem stilisierten Kamerabild zu vergleichen und die Poseberechnung vorzunehmen. Die Pose wird dann an das Video-Widget übergeben, damit dort das Objekt aus der korrekten Pose gerendert werden kann und das Kamerabild überlagert.

Das Darstellen der Kamerabilder in den Widgets geschieht durch Zeichnen eines Polygons in der Größe des Widgets, auf das die Textur des Kamerabildes gemappt wird. Die Initialisierung der Videoquelle durch das ARToolKit geschieht im Video-Widget. Die dabei erstellte Projektionsmatrix muss zusätzlich an das Objekt-Widget übergeben werden, damit das Rendern des Objekts in Video-Widget und Objekt-Widget identisch verläuft.

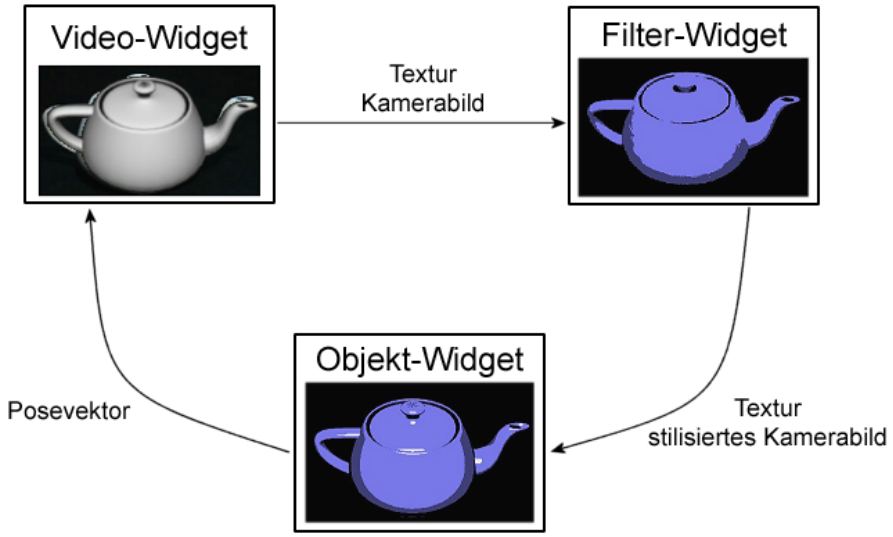

Abbildung 14: Widgets

Signal-Slot-Prinzip Die Verwendung von *Signalen* und *Slots* stellt die Datenkommunikation auf Anwendungsebene zwischen den oben genannten Widgets und mit der GUI her. Für ein Qt-Objekt kann ein Signal definiert werden, das mit dem Aufruf von `emit` ausgelöst wird. Ein Slot ist eine Funktion des empfangenden Qt-Objekts, die bei Eintreffen des Signals ausgeführt wird. Über `connect` wird das Signal eines Qt-Objekts mit einem Slot eines anderen Qt-Objekts verbunden. Den Signalen können Parameter übergeben werden, die an den Slots abgefragt werden. Auf diesem Wege wird etwa der im Objekt-Widget errechnete Posevektor an das Video-Widget gesendet und umgekehrt die Projektionsmatrix bei der Initialisierung an das Objekt-Widget.

Ebenso dient dieses Prinzip der Übergabe von Einstellungen der GUI an die Widgets. So werden der aktuell ausgewählte Stilisierungsfilter und seine eventuelle Stärke, das Ähnlichkeitsmaß und der Schwellwert für die Suchfenstergröße mitgeteilt. Des Weiteren kann die getrackte Pose auf den Initialzustand zurückgesetzt werden und das Logging der Messwerte aktiviert werden. Die Messwerte werden zusätzlich innerhalb der GUI angezeigt (Abbildung 15).

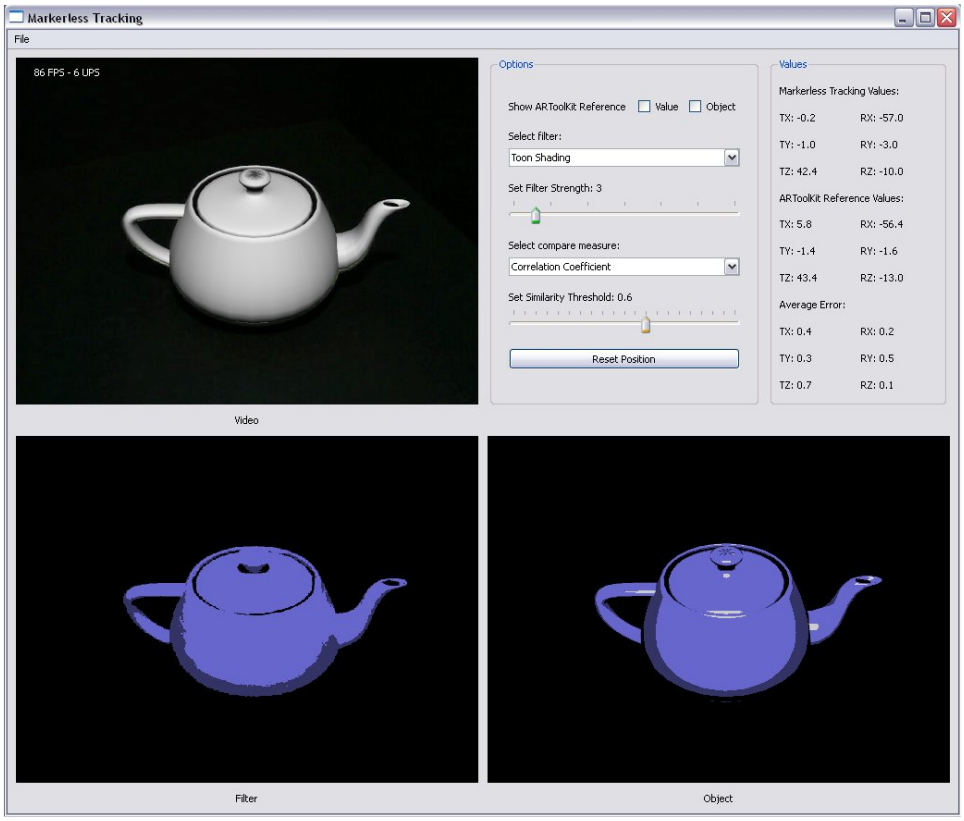

Abbildung 15: GUI, überlagertes Kamerabild (oben), stilisiertes Kamerabild (unten links), stilisiertes Objekt (unten rechts)

6.4 Stilisieren

Die Stilisierung soll sowohl im Video-Widget als auch im Objekt-Widget stattfinden. Daher wird eine gemeinsame Shader-Klasse erstellt, die es ermöglicht, von beiden Widgets aus Shaderdateien zu laden und auszuführen. Die Shader-Klasse und die eingesetzten Shader sind im Anhang A zu finden. Je ein Vertex Shader Programm und ein Fragment Shader Programm ersetzen die feste Funktion der Renderpipeline in OpenGL bezüglich der Verarbeitung von Geometrie-Vertices und Fragmenten. Die Funktion der Shader einzelner Darstellungsformen wurde in Kapitel 5 besprochen. Zur Umsetzung der Stilisierung dient die OpenGL Shading Language (GLSL) [Ros04], welche seit Version 2.0 ein fester Teil von OpenGL mit eigenem Befehlssatz ist. Sie kann auch als Extension[16] verwendet werden.

Die wesentliche Aufgabe der Shader-Klasse wird kurz dargestellt. Das Einbinden der Shader geschieht während der Initialisierung des aufrufenden Widgets. Zuerst werden für Vertex und Fragment Shader je ein Shaderobjekt mit `glCreateShader` definiert. An sie wird mit der Funktion `glShaderSource` der Shader-Quellcode gebunden. Dies geschieht durch Übergabe des Quellcodes in Form eines Pointers auf einen String oder ein Array von Strings. Wenn die Shader in Quelltextdateien vorliegen, muss eine externe Funktion zum Einlesen von Textdateien hinzugezogen werden. Anschließend werden die eingelesenen Shader mit dem Aufruf von `glCompileShader` kompiliert.

Die kompilierten Shadermodule werden dann zu einem lauffähigen Programm gebunden. Dazu wird zunächst mit `glCreateProgram` ein Programmobjekt erstellt, das die Shaderobjekte aufnimmt. Die Shader werden über `glAttachShader` dem Programmobjekt hinzugefügt und das Programm abschließend unter Aufruf von `glLinkProgram` gelinkt.

Der Shader ist jetzt einsatzbereit und kann mit `glUseProgram` vor dem Geometrieaufruf des zu rendernden Objekts aktiviert werden, was im Objekt-Widget das Modell des zu trackenden Objekts und im Video-Widget das Polygon zur Aufnahme der Videotextur sein kann. Um die ursprüngliche OpenGL Renderpipeline zu verwenden, erfolgt die Deaktivierung mit `glUseProgram(0)`. Den Shadern können nach ihrer Aktivierung je nach Variablentyp Einzelwerte, Vektoren und Matrizen übergeben werden. Im vorliegenden Fall soll die Textur mit dem Kamerabild zur Bearbeitung an den Shader übergeben werden. Dies geschieht durch die Angabe der OpenGL Textur-Unit, in der sich die Textur befindet.

[16] GL_ARB_vertex_shader und GL_ARB_fragment_shader

6.5 Ähnlichkeit berechnen

Zur Berechnung der Ähnlichkeit wird das stilisierte Kamerabild aus dem Filter-Widget in das Objekt-Widget übernommen und mit dem dort gerenderten synthetischen Bild verglichen. Da die Berechnung auf der CPU erfolgt, müssen beide Bilder zunächst mit `glReadPixels` in Arrays ausgelesen werden, deren Größe die Bildgröße in Höhe*Breite umfasst. Bei einem dreikanaligen RGB-Farbbild verdreifacht sich die Größe entsprechend. Auf die Bildarrays wird dann in einer Schleife über alle Pixel zugegriffen und die pixelweisen Operationen des gewählten Ähnlichkeitsmaßes ausgeführt. Im Folgenden sind exemplarisch die normalisierte Kreuzkorrelation und der Farbabstand aufgeführt.

```
// Normalisierte Kreuzkorrelation
for (int i = 0; i < width*height; i++){

  sum1 += imageOne[i];
  sum2 += imageTwo[i];
  sum3 += imageOne[i]*imageOne[i];
  sum4 += imageTwo[i]*imageTwo[i];
  corr += imageOne[i]*imageTwo[i];
}

result = (corr-(sum1*sum2)/pixel)
           / sqrt((sum3-(sum1*sum1)/pixel)
                *(sum4-(sum2*sum2)/pixel));


// Farbabstand
for (int i = 0; i < width*height*3; i+=3){

  red     = imageOne[i]  -imageTwo[i];
  green   = imageOne[i+1]-imageTwo[i+1];
  blue    = imageOne[i+2]-imageTwo[i+2];
  length += sqrt(red*red + green*green + blue*blue);
}

result = 1 - length /(width*height*3);
```

Zur Realisierung der statistischen Ähnlichkeitsmaße wurden die Funktionen `cvCreateHist` und `cvCompareHist` der OpenCV [Intel] Bibliothek eingesetzt. Sie sind für die Histogrammerzeugung und den Vergleich zuständig. Die dabei angewendeten Maße sind *Histogramm Intersection*, *Chi-Square-Test* und *Histogram Correlation*.

6.6 Pose generieren

Die Funktion zur Generierung der Parameter für die Kamerapose, aus der die synthetischen Ansichten gerendert werden sollen, läuft wie in Abbildung 13 dargestellt ab. Die Parameter stellen dabei eine Verschiebungs- und Rotationsdifferenz zur letzten korrekten Pose dar, die durch das Suchfenster festgelegt wird. Das Suchfenster besteht wie in Kapitel 3 beschrieben aus 13 Posen. Das Zentrum entspricht mit einer Differenz von Null der zuletzt besten Pose aus dem vorangegangenen Suchfenster und die übrigen werden je nach Suchfenstergröße in Abständen von 2 cm / 1 cm / 0.2 cm bezüglich Translation und 10 Grad / 5 Grad / 1 Grad bezüglich Rotation um das Zentrum erzeugt. Welche Suchfenstergröße gewählt wird, hängt von der Höhe des besten Ähnlichkeitswertes ab. Fällt dieser unter einen Schwellwert, so wird das nächste Suchfenster vergrößert. Steigt die Ähnlichkeit an, werden in einer kleineren Umgebung Posen generiert. Der Schwellwert kann dabei vom Benutzer verändert werden, um die Empfindlichkeit der Beleuchtungssituation anzupassen.

Das von der Aktion *Ähnlichkeit berechnen* gelieferte Ergebnis wird zunächst darauf überprüft, ob es der im aktuellen Suchfenster beste Ähnlichkeitswert ist. In diesem Fall, wird die Ähnlichkeit und die dazugehörigen Differenzparameter für den Posevektor gespeichert. Ansonsten werden sie verworfen. Danach erfolgt die Prüfung, ob das Suchfenster bereits abgearbeitet wurde. Ist dies nicht der Fall, wird durch Veränderung der Differenzparameter im Posevektor eine neue Pose des aktuellen Suchfensters erstellt und als *Nächste Pose* zur Erzeugung des nächsten synthetischen Bildes ausgegeben.

Das Suchfenster gilt als vollständig, wenn alle 13 Posen generiert wurden. Nun werden diejenigen Differenzparameter, welche unter den 13 getesteten dem höchsten Ähnlichkeitswert zugeordnet sind, auf die letzte korrekte Pose aufaddiert. Hat keine Bewegung stattgefunden, bleibt die vorhergehende Pose durch eine Differenz von Null unverändert. Die angepasste Pose wird als *Beste Pose* an das Video-Widget gesendet, damit dort das Objekt über das Kamerabild gerendert werden kann. Gleichzeitig erfolgt die Festlegung der Größe des nächsten Suchfensters und die Differenz zur Bildung dessen erster Pose wird als *Nächste Pose* ausgegeben.

6.7 Versuchsaufbau

Da bereits das Ansteuern der Videoquelle über die Funktionen des ARToolKit erfolgt, liegt es nahe, dessen Trackingbibliothek zur Messung der Genauigkeit des eigenen markerlosen Trackings einzusetzen. Das Trackingmodul bietet die Möglichkeit, vordefinierte Marker im Kamerabild zu erkennen und deren Pose bis in den Millimeterbereich genau zu bestimmen. Es liefert also geeignete Referenzwerte, deren Differenz zu den Daten des

eigenen Ansatzes ein gutes Maß für die Beurteilung der Genauigkeit darstellt. Dazu wird ein Versuchsaufbau gewählt, bei dem das zu trackende Objekt in einem festen, bekannten Abstand zu dem Marker platziert wird. Die Referenzwerte werden bei der Messung um diesen Abstand korrigiert, damit die Mittelpunkte von Marker und Objekt zusammenfallen.

Abbildung 16: Versuchsaufbau

Für eine exakte Messung ist es notwendig, dass die verwendete Kamera mit den Methoden des ARToolKit kalibriert wird. Die daraus resultierenden intrinsischen Kameraparameter werden zur Herleitung einer OpenGL-Projektionsmatrix verwendet, die zur Grundlage der Messung dient. Da die Werte des ARToolKit jedoch in Millimetern gemessen werden, die Transformation eines Objekts aber in OpenGL Einheiten erfolgt, muss zunächst eine Skalierung vorgenommen werden, die einen Zentimeter in der Realität auf eine OpenGL Einheit abbildet. Dazu stellt das ARToolKit die Funktion `arglCameraViewRH` zur Verfügung, die den Skalierungsfaktor mit der Transformationsmatrix verrechnet. Als weiterer Schritt ist eine Mittelung der Messwerte des ARToolKit über 100 Frames notwendig, da diese unter starkem Rauschen leiden.

Das ARToolKit bestimmt die Pose des Markers in Kamerakoordinaten. Zur Vereinfachung wird auch im markerlosen Ansatz die Kamera im Ursprung des Koordinatensystems belassen und das Objekt transformiert. Die so einheilichen Werte für die Pose des Markers, des Objekts und deren Differenz werden über die GUI ausgegeben und zusätzlich in einem Logfile festgehalten.

Der markerbasierte Trackingvorgang, wie er im ARToolkit stattfindet, verläuft kurz beschrieben wie folgt. Bei der Initialisierung wird ein Muster des zu detektierenden Markers mit `arLoadPatt` geladen. Das von der

Videoquelle bezogene Bild liegt zunächst in durch die Kameraoptik verzerrter Form vor. Es muss zuerst über die per Kamerakalibrierung ermittelten und extern gespeicherten intrinsischen Kameraparameter entzerrt und in ideale Bildschirmkoordinaten gebracht werden. Danach folgt mit `arDetectMarker` die Erkennung des Markers im Bild. Dies beinhaltet die Umwandlung in ein schwarz-weißes Binärbild, welches mit Hilfe von Schwellwertverfahren und Bildverarbeitungsalgorithmen zur Eckendetektion auf die quadratischen schwarzen Strukturen der Marker geprüft wird. Das Muster im Inneren des Markers wird in einem Pattern-Matching Verfahren mit gespeicherten Mustern verglichen. So kann der Marker eindeutig identifiziert und ihm ein zu renderndes Objekt fest zugewiesen werden. Dieses Vorgehen ermöglicht den simultanen Einsatz mehrerer Marker.

Anhand eines gefundenen Markers werden mit `arGetTransMat` die Position und Orientierung des Markers bezüglich der Kamera berechnet. Dazu kommen Algorithmen aus der Disziplin des Rechnersehens zum Einsatz. Sie leiten über die im aktuellen Frame gefundenen Merkmale und durch Vergleich mit dem vorhergehenden Frame die Transformationsparameter her, welche die Lage des Markers in 3D-Koordinaten von der Kamera aus gesehen bestimmen. Sie werden als Transformationsmatrix an OpenGL übergeben und dienen der korrekten Translation und Rotation gerenderter Objekte.

7 Ergebnisse

Im ersten Abschnitt dieses Kapitels soll dargelegt werden, welche Ähnlichkeitsmaße für das Tracking als geeignet erscheinen. Dazu wird das Verhalten der Ähnlichkeitsmaße bei Rotation und Translation eines Objekts bezüglich eines Referenzbildes untersucht. Anschließend werden Kombinationen von Ähnlichkeitsmaßen und Arten der Bildstilisierung auf die Aspekte Geschwindigkeit und Genauigkeit im Trackingeinsatz getestet.

7.1 Analyse der Ähnlichkeitsmaße

Zur Untersuchung der in Kapitel 4 beschriebenen Ähnlichkeitsmaße wird aus dem grafischen Modell eines Objekts ein Referenzbild erzeugt, welches die Referenzansicht im Nullpunkt darstellt. Anschließend wird das Objekt von dieser Ansicht aus in der ersten Messung in Schritten von 0.2 OpenGL Einheiten in der XY-Ebene transliert. In der zweiten Messung findet eine Rotation in Abständen von 0.5 Grad um die X- und Y-Achse statt. Zu jeder Objektbewegung wird eine neue Ansicht gerendert, die mit dem Referenzbild verglichen wird. Die sich dabei ergebenden Ähnlichkeitswerte sind in den folgenden Diagrammen auf der Z-Achse dargestellt, die entsprechende Objektbewegung auf der X- und Y-Achse.

Als ideales Maß wird im Hinblick auf das Optimierungsverfahren ein glockenförmiger Verlauf der Ähnlichkeitswerte erwartet, welcher im Mittelpunkt des Diagramms, also bei vollkommener Übereinstimmung von Referenzbild und gerenderter Ansicht, ein möglichst ausgeprägtes Maximum hat und mit zunehmender Abweichung von der Referenzansicht stärker abfällt. Benachbarte Werte sollten nicht zu dicht beieinander liegen, aber auch keine zu großen Differenzen beinhalten, da dies zu unruhigem Tracking (Jittering) oder gar dem Verlust des getrackten Objektes führt. Die Ähnlichkeitswerte sind der besseren Vergleichbarkeit wegen so normiert und gegebenenfalls invertiert, dass sie in einem Intervall zwischen 0 und 1 liegen, wobei 0 die geringste und 1 die größtmögliche Ähnlichkeit bezeichnen.

Die auf den Intensitätswerten des Bildes basierenden Ähnlichkeitsmaße *Sum of Squared Differences* (SSD), *Cross Correlation* (CC) und *Normalized Cross Correlation* (NCC) werden zunächst auf dem Standardbeleuchtungsmodell der OpenGL Pipeline mit diffusem Shading getestet. Alle drei zeigen hierbei sowohl bezüglich der Translation als auch der Rotation das gewünschte Verhalten. Besonders bei der Translation ist der Werteverlauf als nahezu ideal zu bezeichnen. Bei der Rotation ist vor allem der Bereich um das Optimum interessant, da bei den für die Optimierung relevanten kleinen Bewegungen ausreichend große messbare Veränderungen in den Ähnlichkeitswerten generiert werden.

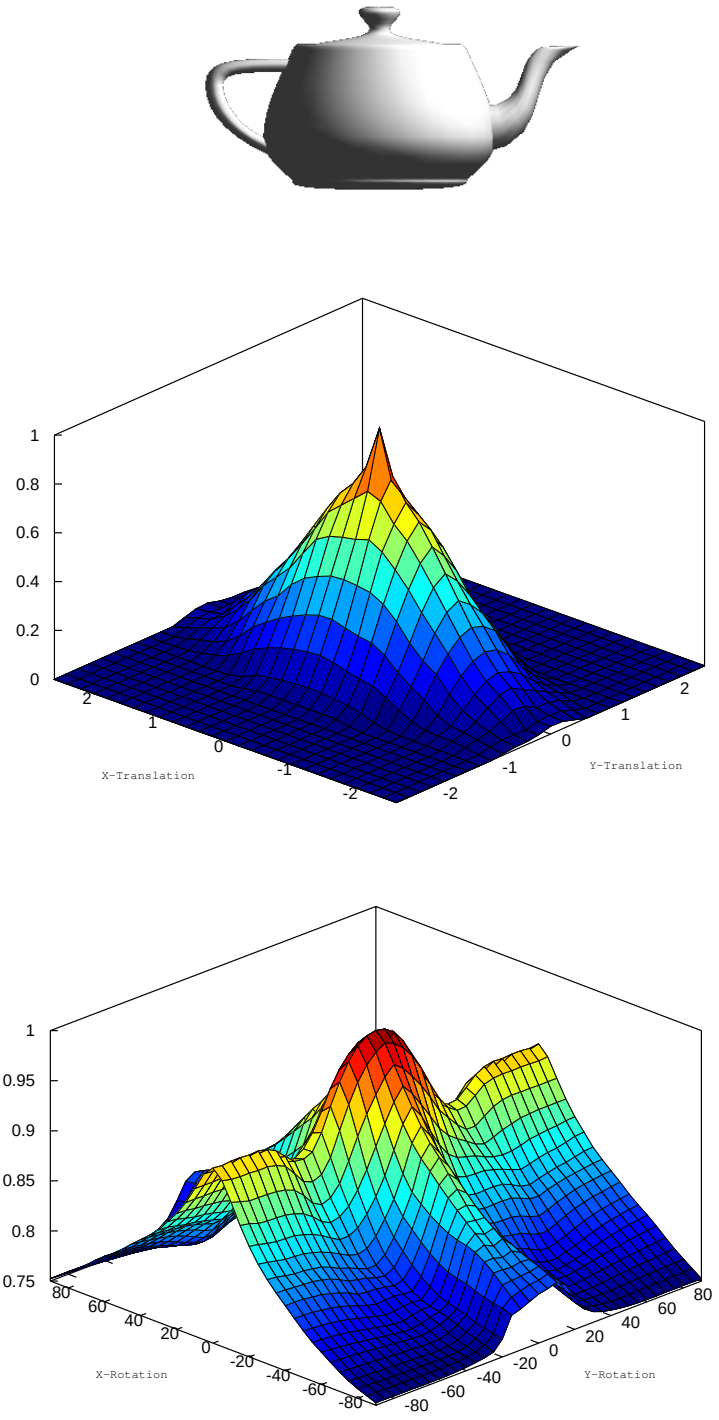

Abbildung 17: Normalized Cross Correlation: Referenzbild, Translation, Rotation

Die drei getesteten Maße unterscheiden sich im Vergleich nicht wesentlich in ihrem Verhalten (vgl. Abbildung 17 der Normalisierten Kreuzkorrelation und Abbildung 18 der Summe der Quadratischen Differenzen). Unterschiede sind lediglich in den Intervallen zu erkennen, in denen die jeweiligen Ergebniswerte liegen. Im realen Trackingeinsatz mit einer Kamera zeigt sich jedoch, dass die NCC durch ihre Mittelwertfreiheit einen entscheidenden Vorteil gegenüber SSD und CC hat.

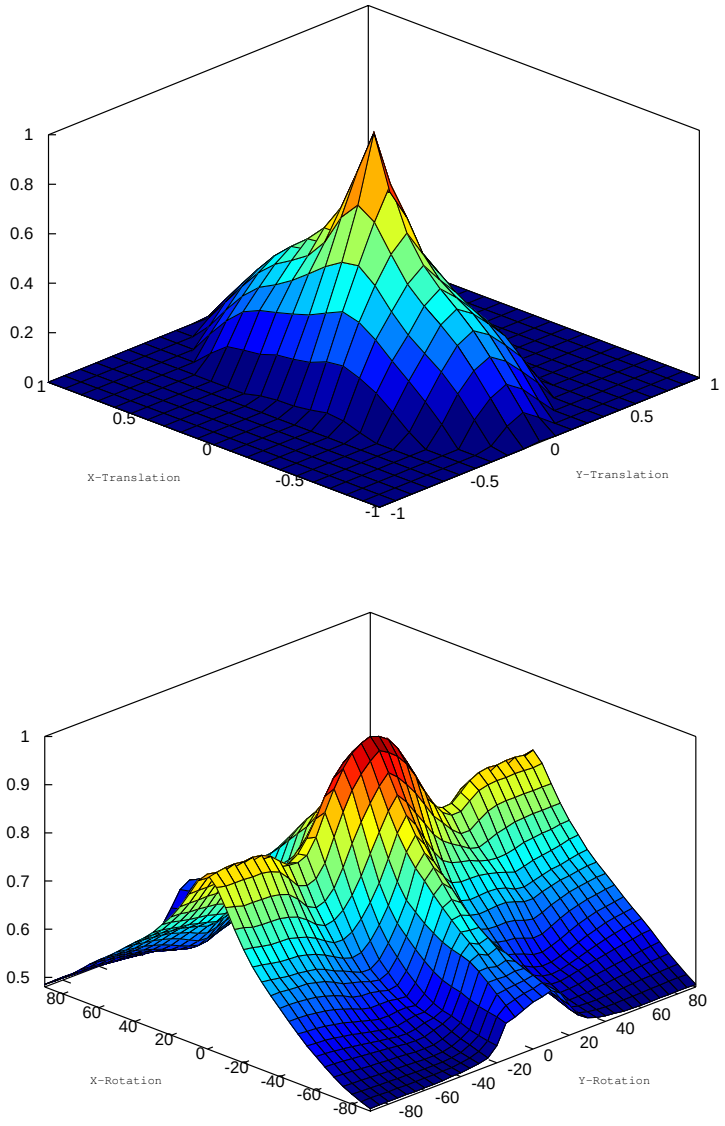

Abbildung 18: Sum of Squared Differences

Helligkeitsschwankungen werden durch die NCC gefiltert, welche bei den anderen Maßen einen großen Einfluss auf die Stabilität der Werte haben und teils zu Werteverschiebungen führen. Die bisher betrachteten Ähnlichkeitsmaße basierten auf den reinen Helligkeitswerten der Pixel. Dem soll in Abbildung 19 mit der Color Difference (CD) der einfache Abstand der Farbvektoren als farbbasiertes Maß gegenüber gestellt werden.

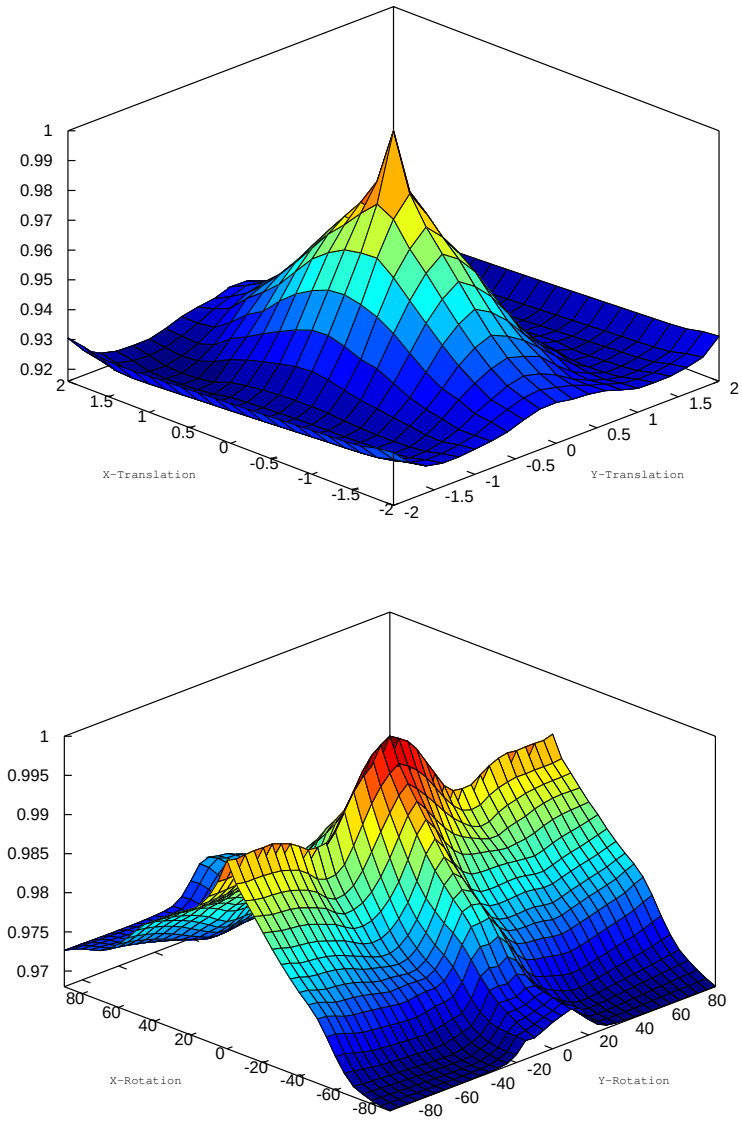

Abbildung 19: Color Difference

Hier ist eine Stauchung des Wertebereichs zu beobachten. Da nun mit den drei Farbkanälen mehr Größen Einfluss auf die Berechnung des Ähnlichkeitswertes haben, ist mit einer erhöhten Störanfälligkeit zu rechnen. Auch der schärfere Grat entlang der Maximumslinie des Werteverlaufs lässt eine Werteunruhe (Jittering) erwarten, die zu unpräzisen Ergebnissen in der Kamerapose führen kann.

In den Abbildungen 20 und 21 wird die NCC auf Ansichten getestet, die mit einem Gooch-Shader und einem Toon-Shader stilisiert wurden. Die Ergebnisse lassen jedoch nicht die erhoffte signifikante Verbesserung des Verhaltens der Ähnlichkeitsmaße erkennen, welche einem robusteren Tracking zugute kommen würde. Im Falle des Toon-Shadings zeigen sich sogar wieder stärkere Grate, die die Empfindlichkeit gegenüber Störungen erhöhen und unpräzises Tracking fördern.

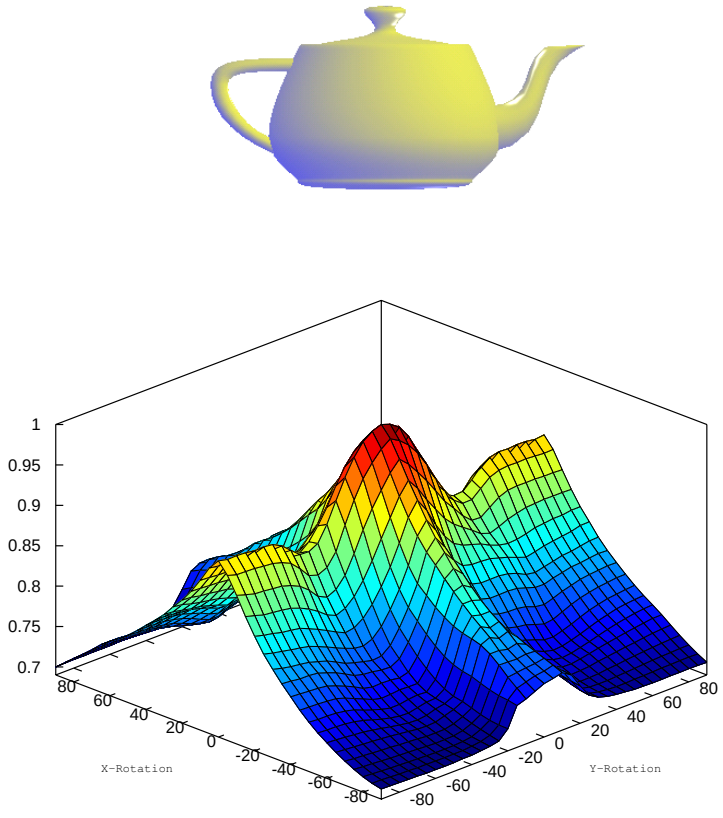

Abbildung 20: NCC unter Gooch-Shading

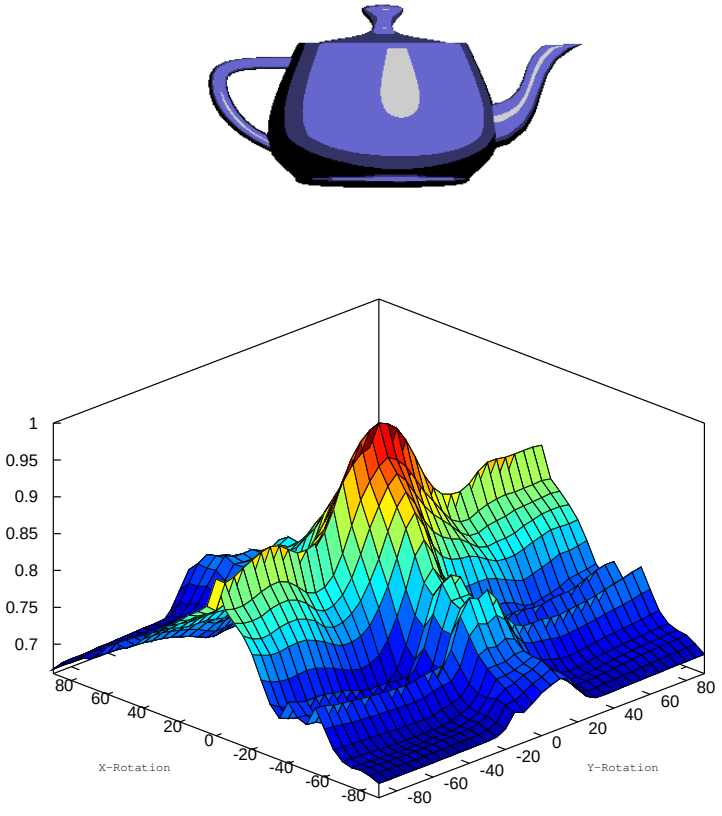

Abbildung 21: NCC unter Toon-Shading

In Abbildung 22 wird die NCC auf mit dem Sobelfilter erstellte Kantenbilder angewandt. Durch den geringen Anteil der Kantenpixel am Gesamtbild sinkt die Ähnlichkeit im Vergleich zu anderen Darstellungsformen. So entstehen bereits bei kleinen Bewegungen des Objekts extreme Werteunterschiede durch stark abfallende Flanken entlang des Wertemaximums. Dies legt die Vermutung nahe, dass das Tracking des Objekts bei schnellen Bewegungen leicht verloren gehen könnte.

Dem kann jedoch entgegengewirkt werden, indem die Kantenbreite verstärkt wird und der Bereich, in dem Übereinstimmungen zwischen Referenz- und Vergleichsbild gefunden werden können, vergrößert wird. Der Sobelfilter auf einem zuvor mittelwertgefilterten Bild führt dazu, dass die Werte um das Maximum nicht mehr so stark abfallen (Abbildung 23). Damit erhöht sich auch die Präzision des Maßes bei kleinen Bewegungen, vor allem bei Betrachtung der Rotation. Der zuvor extreme Peak bietet nun eine breitere Basis für die Optimierung.

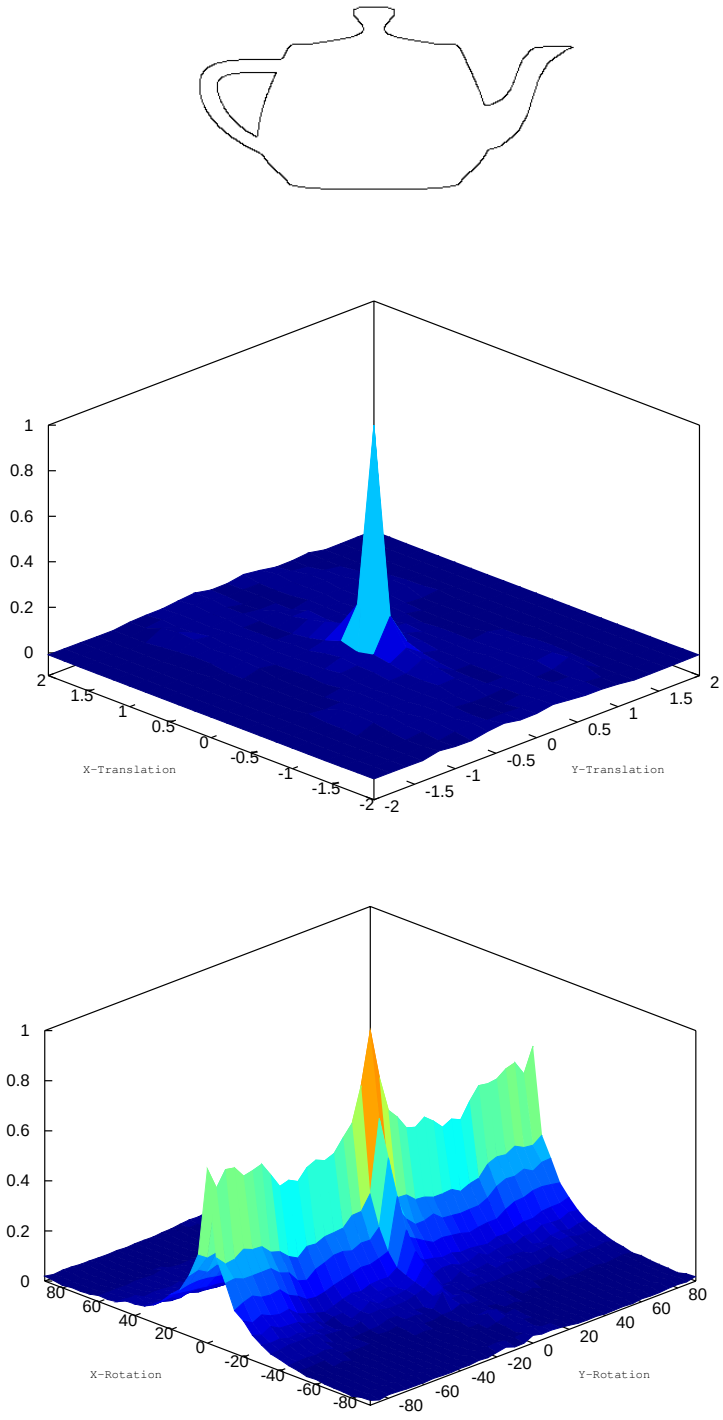

Abbildung 22: NCC auf Sobel-Kanten: Referenzbild, Translation, Rotation

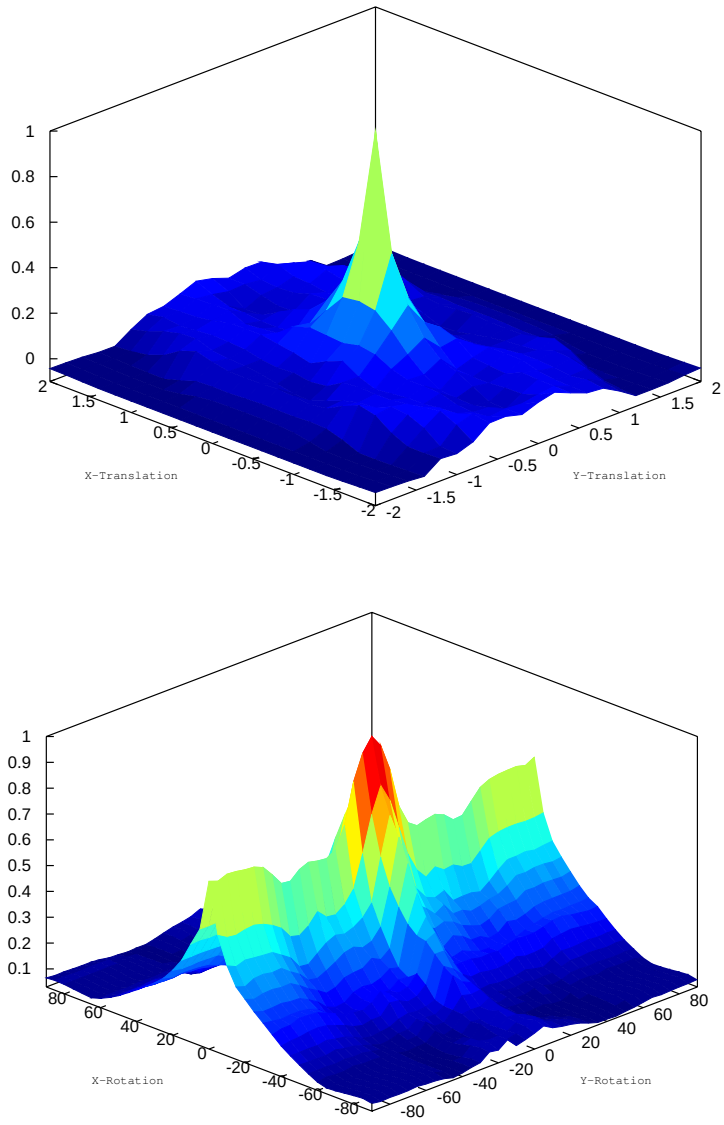

Abbildung 23: NCC auf Sobel-Kanten, stärkere Linien

Neben den direkt-intensitätsbasierten Ähnlichkeitsmaßen werden auch statistische Maße getestet, die auf der vorherigen Berechnung eines Histogramms beruhen. Hierbei werden sowohl Farbhistogramme zum Vergleich herangezogen, als auch Histogramme über die Häufigkeitsverteilung der Gradientenwinkel bei der Sobelfilterung. Die Gradientenwinkel eignen sich nicht als Ähnlichkeitsmaß, da sie kein Maximum bezüglich der Translation ausbilden. Wird das Objekt transliert, ändern sich die Winkel der Gradienten kaum.

Im Test wurden die Maße *Histogram Intersection*, *Chi-Square-Test* und *Histogram Correlation* verglichen. Unter Verwendung von Farbhistogrammen zeigt sich eine leichte Steigung des Werteverlaufs zum Maximum hin. Bei der Translation fällt der Werteanstieg jedoch nicht deutlich genug aus, um das Auffinden des Maximums zu gewährleisten.

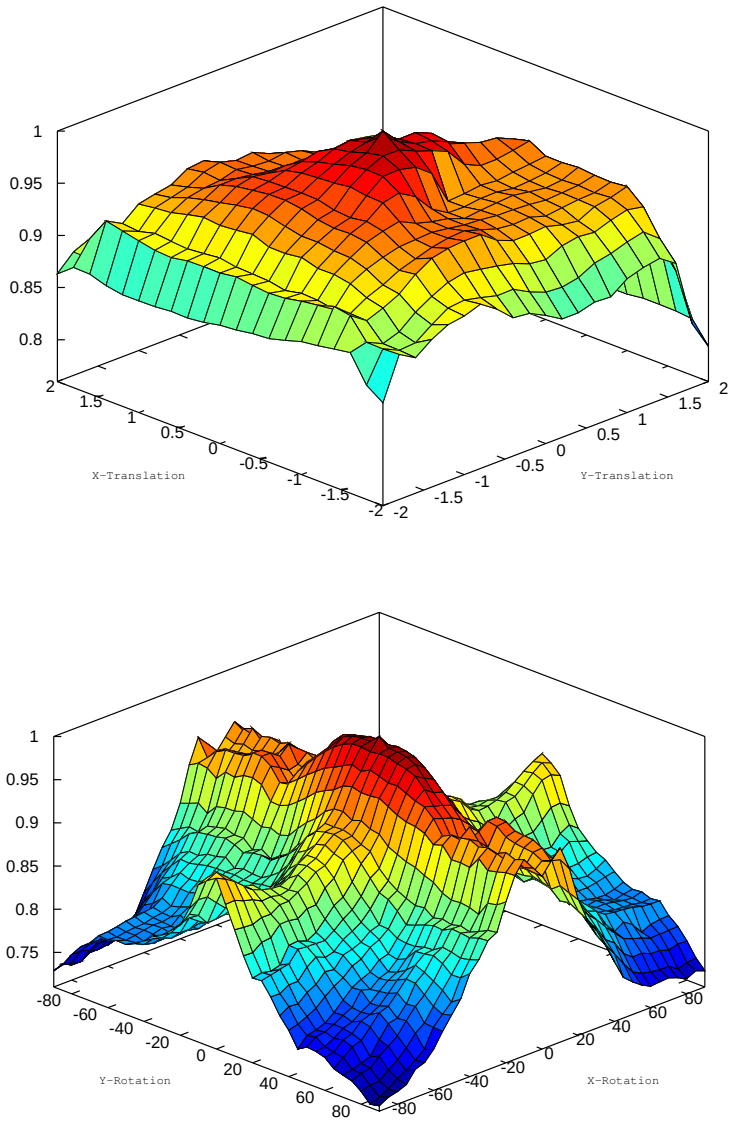

Abbildung 24: Histogram Intersection

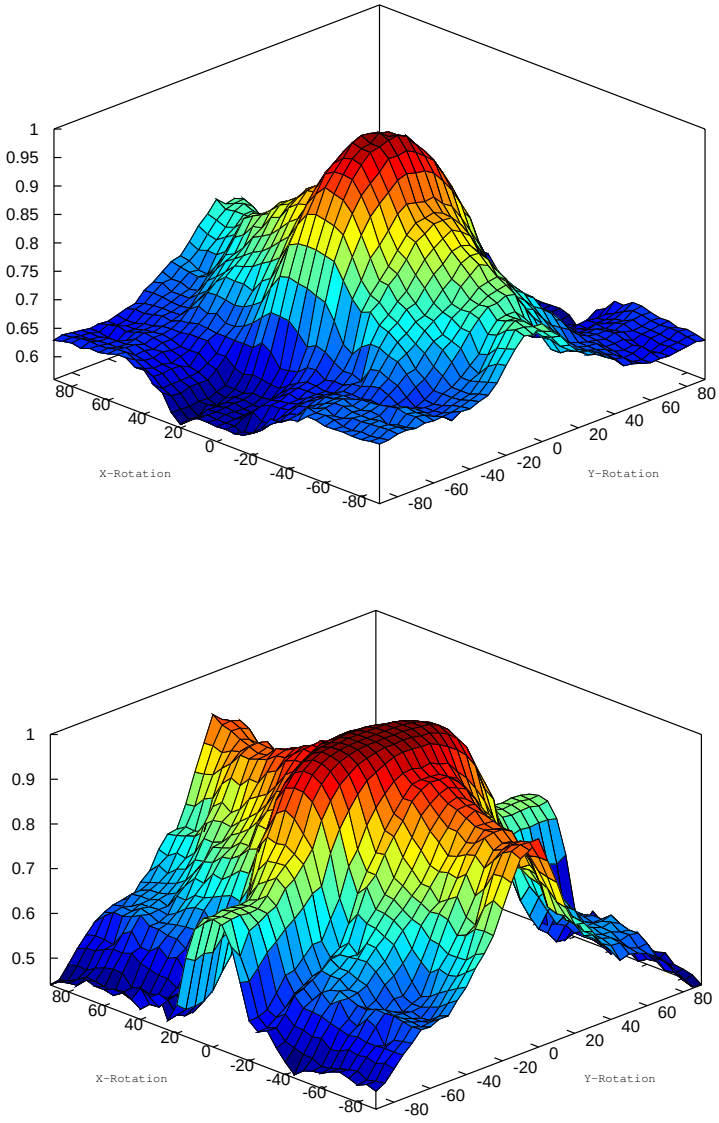

Abbildung 25: Histogram Chi-Square-Test, Histogram Correlation

Auch bei kleinen Rotationen sind die Werteänderungen nicht ausreichend, da die Maße sehr flache Werteebenen im Maximum haben. Allein die Anwendung des Chi-Square-Tests zeigt bei der Rotation ein annähernd gutes Ergebnis.

Als Zwischenergebnis kann festgehalten werden, dass die Normalisierte Kreuzkorrelation aufgrund ihrer Robustheit gegen Intensitätsveränderungen im Bild am geeignetsten für den Einsatz als Ähnlichkeitsmaß im

Tracking anzusehen ist. Generell scheint das Hinzufügen von Strukturen durch starke Unterschiede zwischen hellen und dunklen Bereichen, durch Farbvereinfachung und durch Kantendetektion die Genauigkeit nicht zu erhöhen und kann sogar negative Auswirkungen in Form von Jittering haben. Es zeigt sich in diesem Zusammenhang auch eine besondere Abhängigkeit von der Lichtquelle. Nur wenn diese entsprechend der realen Lichtquelle korrekt positioniert ist, kann die gesteigerte Empfindlichkeit der Ähnlichkeitsmaße, wie sie durch Selbstverschattung oder Toon-Shading entsteht, genutzt werden, höhere Präzision im Tracking zu gewinnen. Im Gegenzug kann Reduktion der Struktur des realen Objekts durch einen leichten Blur des Kamerabildes die Ähnlichkeit nur minimal anheben.

In Tabelle 1 werden die maximal erreichten Ähnlichkeitswerte der Maße in Kombination mit verschiedenen Shadings verglichen. Für die Sobel-Filterung sind jeweils ein Wert für geringe und höhere Kantenstärke aufgeführt. Im nächsten Abschnitt ist die NCC wegen ihrer Robustheit ein Kandidat für weitergehende Untersuchungen. Daneben wird auch die Color Difference berücksichtigt, da sie wie aus der Wertetabelle zu entnehmen ist, auf Vorteile bei Shading mit einheitlicher Farbgebung hoffen lässt.

	Diffuses Shading	Selbstver-schattung	Toon Shading	Gooch Shading	Sobel
NCC	0.96	0.97	0.95	0.93	0.61/0.70
CC	0.96	0.96	0.96	0.94	0.63/0.71
SSD	0.92	0.93	0.92	0.87	0.33/0.43
CD	0.97	0.97	0.99	0.98	0.97/0.98

Tabelle 1: Vergleich der Ähnlichkeitsmaß-Maxima

Ein Problem bei der Verwendung intensitätsbasierter Ähnlichkeitsmaße ergibt sich aus der Gesamthelligkeit des Bildes. Wird die Intensität etwa durch Fremdobjekte im Bild erhöht, nimmt der Anteil des zu trackenden Objekts am Ergebnis des Ähnlichkeitswertes ab und die maximale gemessene Ähnlichkeit sinkt. Die Optimierung ist jedoch von der aktuell gemessenen Ähnlichkeit abhängig, da diese als Schwellwert für die Wahl der Suchraumgröße dient, was zu einer starken Beeinflussung des Ergebnisses durch Fremdobjekte führt. Bei beliebigem Hintergrund erhöht sich die Gesamthelligkeit und der Schwellwert für den Suchraum der Optimierung muss nachjustiert werden. Eine Figur-Grund-Trennung per Stencil-Buffer zur Ausmaskierung störender Bildbereiche und eine Gewichtung der relevanten Werte im Bild brachten nicht den gewünschten Effekt. Daher ist es im gewählten Ansatz nötig, die Versuche auf einem standardisierten Hintergrund durchzuführen, welcher mit demjenigen aus dem synthetisch erzeugten Vergleichsbild übereinstimmt.

7.2 Qualität des Trackings

Im Folgenden soll der gewählte Tracking-Ansatz durch die Messung des Fehlers bezüglich eines Referenzkoordinatensystems bewertet werden. Die Referenzwerte liefert das markerbasierte Tracking des ARTooKits [ArLib]. Da dessen Werte einem Rauschen unterliegen (Abbildung 26), werden sie gemittelt. Der gemessene Fehler ergibt sich aus den Differenzen zwischen den Koordinaten des ARToolKits und der aktuell generierten Pose.

7.2.1 Geschwindigkeit

Das System liefert maximal 100 Frames pro Sekunde, in denen die Darstellung der Kamerabilder erfolgt. Für ein Kamerabild werden wie in 6.6 beschrieben 13 synthetische Vergleichsbilder gerendert. Der Rendervorgang des synthetischen Bildes erfolgt in einem OpenGL Kontext, der parallel zum Auslesen der Kamerabilder läuft. Daher wird nach jeweils 13 Kameraframes die optimierte Pose eines Suchfensters erzeugt, was unter Berücksichtigung von Framerateschwankungen bei der Anwendung verschiedener Filter und Maße für das Tracking eine Geschwindigkeit von 5-7 Frames pro Sekunde ergibt, mit denen das Kamerabild mit dem aktualisierten virtuellen Objekt überlagert wird.

7.2.2 Genauigkeit verschiedener Shadings

Die folgenden Diagramme zeigen die Verwendung von NCC und CD auf verschiedenen Darstellungsformen der Vergleichsbilder. Es werden jeweils ein Versuch mit entsprechend der realen Lichtquelle korrekt positionierter virtueller Lichtquelle durchgeführt und ein Versuch mit suboptimaler Lichtquelle. In Ruhelage ohne Bewegung werden die Shadings in dieser Reihenfolge aktiviert: Frame 0-1000 Diffuses Shading, Frame 1000-2000 mit Selbstverschattung, Frame 2000-3000 Toon Shading, Frame 3000-4000 Gooch Shading, Frame 4000-5000 Sobel Kantenbild. Die Anzahl der Frames ist auf der X-Achse aufgetragen, der Fehler in cm oder Grad auf der Y-Achse.

Diffuses Shading Zeigt im Vergleich mit den anderen Darstellungsformen das stabilste Verhalten mit durchschnittlichem Fehler von 0.1 cm und 0.3 Grad. Auch bei inkorrekter virtueller Beleuchtung werden noch die besten Werte in Translation und Rotation erreicht.

Schatten Das Hinzufügen von Selbstverschattung kann bei korrekter Beleuchtung zur Stabilisierung der Rotationswerte führen und den Rotationsfehler um 1-2 Grad verringern. Bei schlechter Lichtquellenposition kommt

es aber hingegen zur Verschlechterung der Translation und Rotation gegenüber dem diffusen Shading ohne Schatten.

Toon Shading Führt bei korrekter Beleuchtung zu genauerer Translation im Bereich der Tiefenwerte auf der Z-Achse, die Rotation verschlechtert sich jedoch geringfügig. Diese Darstellungsart zeigt sich bei schlechter Beleuchtung als besonders empfindlich in der Z-Translation und erzeugt hier mit einer Abweichung von 0.75 cm den größten Fehler.

Gooch Shading Im Vergleich am ungeeignetsten für das Tracking, da mit einer Abweichung von 0.45 cm und 1.6 Grad der größte Grundfehler in Translation und Rotation bei Ruhelage erreicht wird. Es reagiert insgesamt am empfindlichsten auf von der Realität abweichende Beleuchtung mit einem Fehler von 3.1 Grad in der Rotation.

Sobel Filter Der Vergleich von Kantenbildern erzeugt Translationsfehler auf dem relativ guten Niveau des Toon Shadings. Die Werte der Rotation sind jedoch weniger stabil und führen unabhängig von der Beleuchtung zu höherem Jittering.

	Diffuses Shading	Selbstver schattung	Toon Shading	Gooch Shading	Sobel
NCC Translation (cm)	0.25	0.35	0.75	0.25	0.35
NCC Rotation (Grad)	0.35	1.2	1.8	3.1	2.3
CD Translation (cm)	0.2	0.2	0.1	0.1	0.1
CD Rotation (Grad)	0.5	0.5	1.0	2.0	1.0

Tabelle 2: Lichtquellenfehler

In Tabelle 2 sind dazu die maximalen Fehler aufgeführt, die bei der jeweiligen Darstellung durch inkorrekte Beleuchtung entstehen. Diese führte in den Versuchen mit der NCC zu Abweichungen bis zu 0.75 cm und 3.1 Grad, welche sich besonders auf die Tiefenwerte und Rotation um die X-Achse auswirken. Bei der CD fällt der Einfluss der Lichtquelle nicht ganz so stark aus. Sie zeigt sich größtenteils weniger empfindlich gegenüber inkorrekter Beleuchtung. Da sie jedoch bereits in Ruhe und bei korrekter Beleuchtung mit Abweichungen zur NCC von bis zu 0.5 cm und 4 Grad einen wesentlich größeren Grundfehler erzeugt (vgl. Abbildung 27 und 29), der

vor allem bei der Rotation zum Tragen kommt, ist sie ungeeigneter für genaues Tracking. Des Weiteren tritt besonders im Bereich der Tiefenwerte verstärktes Jittering auf. Es ist zwar eine Stabilisierung der Werte bei Einsatz von farbbasiertem Shading zu erkennen, doch eine sehr gute Verbesserung des Translationsfehlers geht besonders bei Gooch Shading mit einer starken Verschlechterung des Rotationsfehlers einher. Damit bringt die CD keinen nennenswerten Vorteil gegenüber der NCC, weshalb in den folgenden Untersuchungen des Trackings in Bewegung nur noch die NCC zum Einsatz kommt.

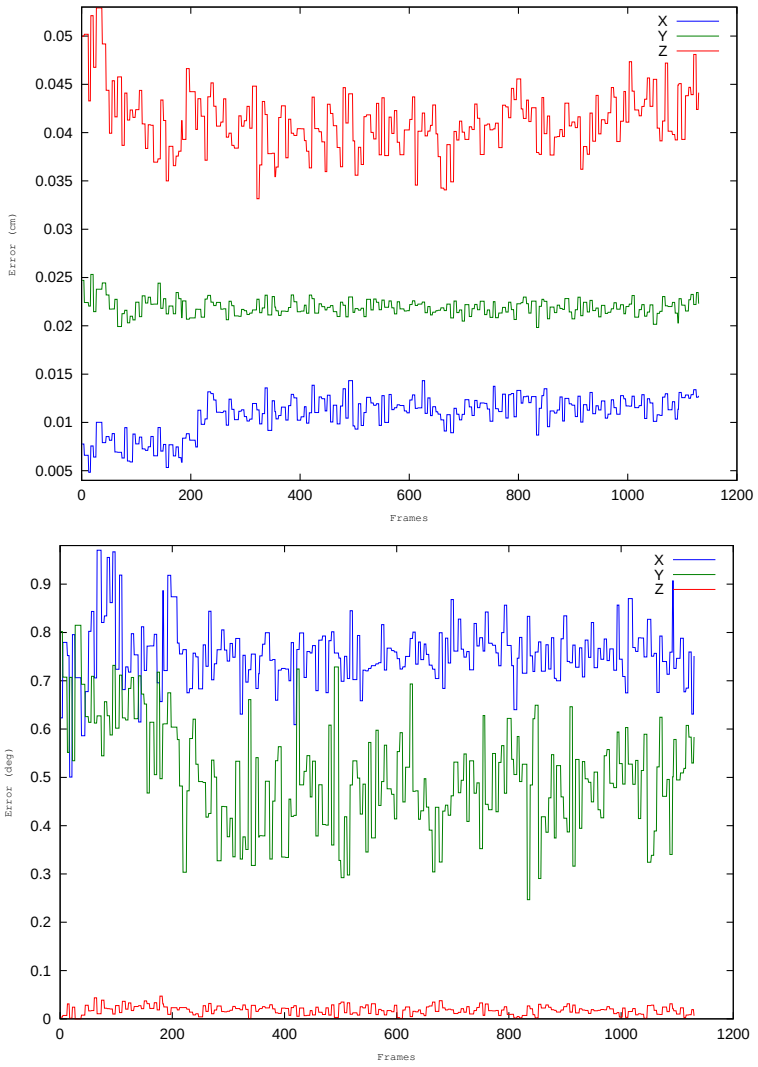

Abbildung 26: ARToolKit Rauschen

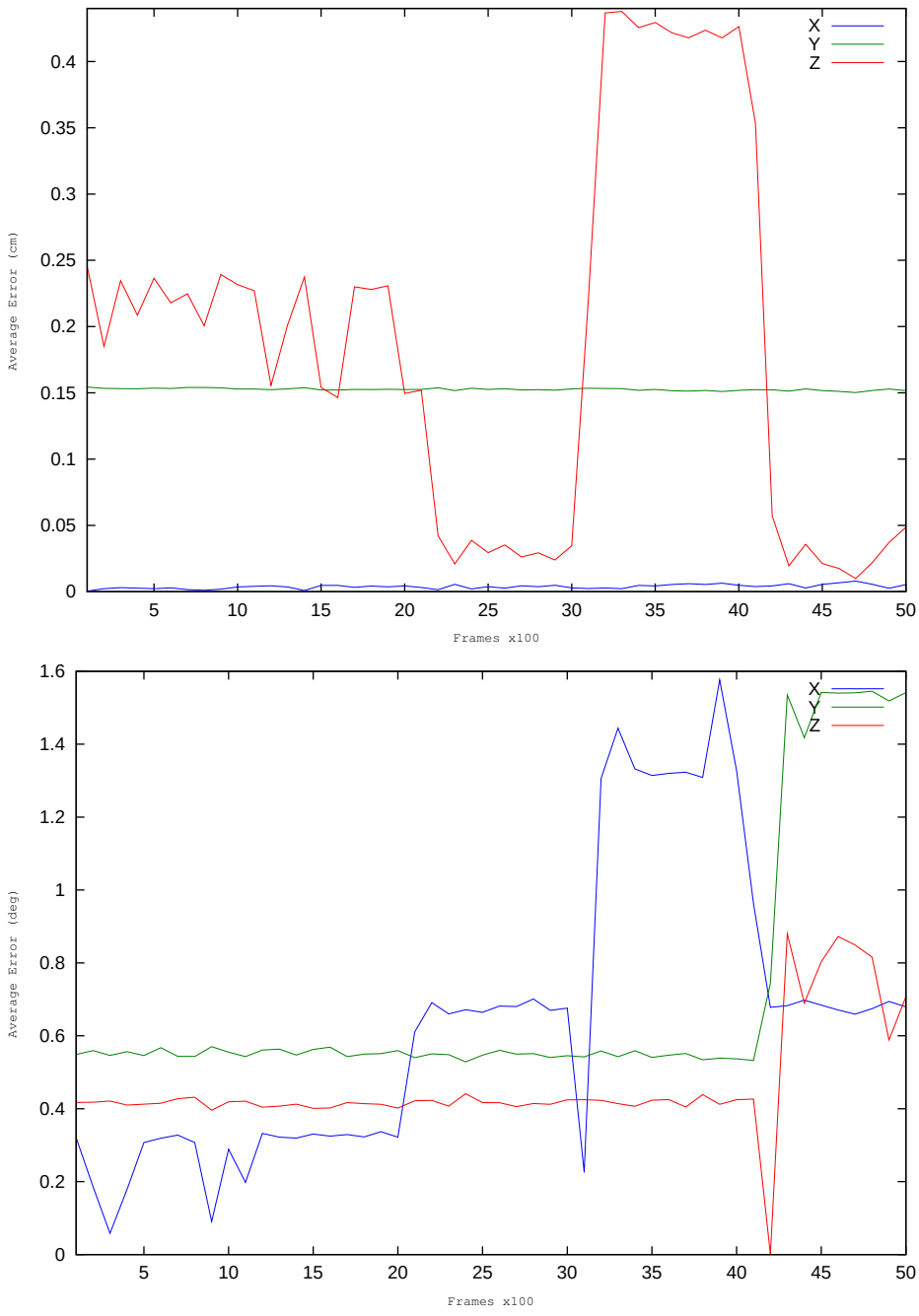

Abbildung 27: NCC, korrekte Lichtquelle

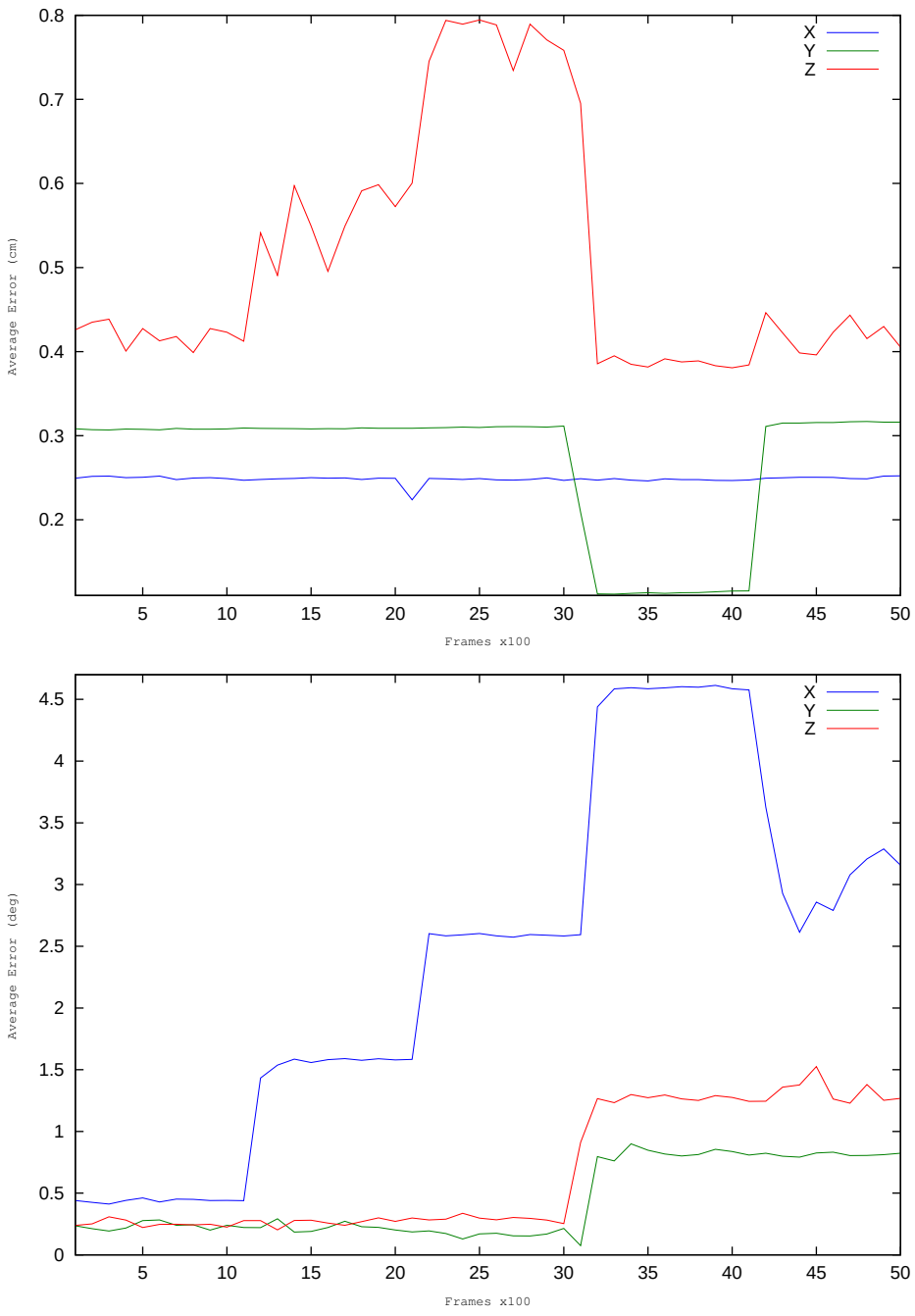

Abbildung 28: NCC, nicht optimale Lichtquelle

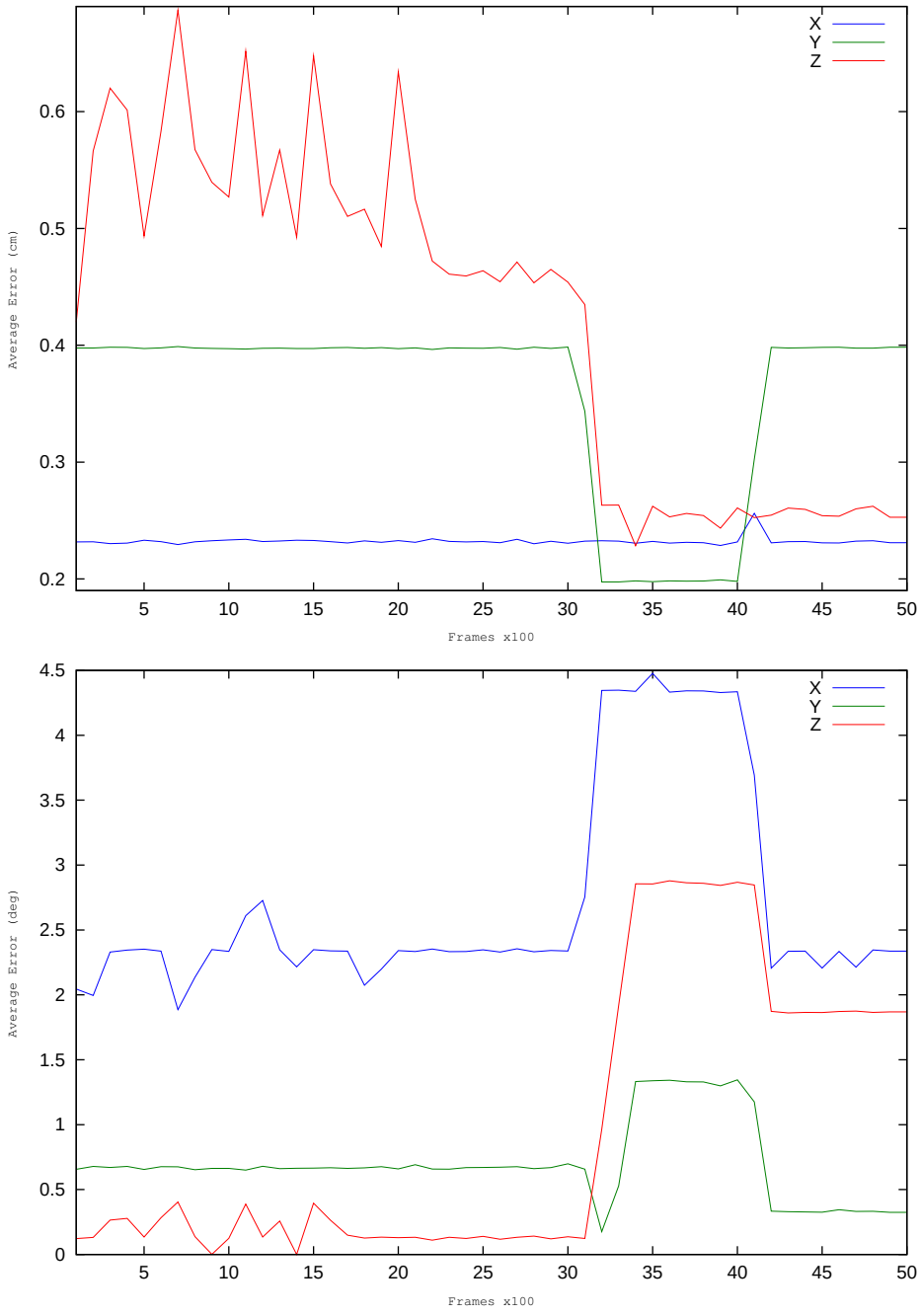

Abbildung 29: CD, korrekte Lichtquelle

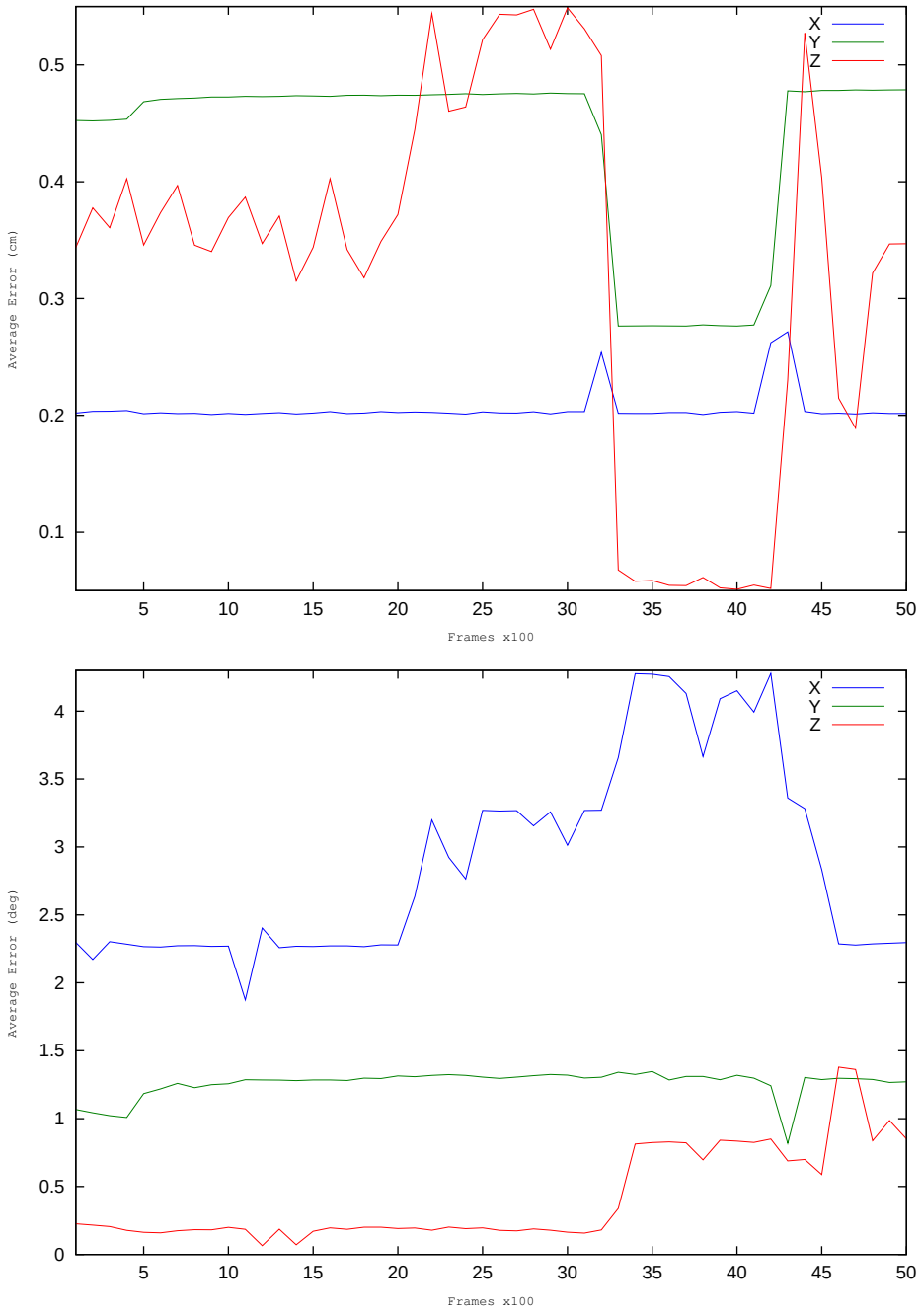

Abbildung 30: CD, nicht optimale Lichtquelle

7.2.3 Stabilität und Angleichung bei Bewegung

Zur Bestimmung der Genauigkeit des Trackings unter Bewegung wird zunächst die Abhängigkeit des Fehlers von dem Suchfenster der Optimierung getestet. In Abbildung 31 wird ab Frame 500 eine kontinuierlich langsame Bewegung entlang der Y-Achse ausgeführt. Dabei wird das Suchfenster von der Optimierung klein gewählt, weil der Schwellwert zum Wechseln in den nächstgrößeren Suchbereich nicht überschritten wird. Der Translationsfehler bleibt während der Bewegung unter 0.5 cm. Im Bereich Frame 2500 wird eine schnelle Bewegung ausgeführt, die bei größerem Suchfenster einen Fehler von 3.5 cm in Translationsrichtung verursacht. Die Ausgangspose wird danach mit minimal größerem Fehler wieder eingenommen.

In Abbildung 32 wird in Frame 500-1000 eine kontinuierliche Rotation um die Z-Achse durchgeführt. Bei kleinem Suchfenster beträgt der Winkelfehler nicht mehr als 2 Grad. Im Bereich Frame 1500 folgt eine schnelle Rotation, unter dessen vergrößertem Suchfenster der Fehler auf 16 Grad anwächst, aber nach der Angleichung wieder bis auf den Ausgangsfehler kleiner 2 Grad abfällt.

Um die mittlere Zeit zu bestimmen, die das Tracking benötigt um nach einer Bewegung wieder eine stabile Pose zu generieren, wird nun der Abstand der Kamera zum Objekt entlang der Z-Achse vergrößert und dabei Translations- als auch Rotationsfehler gemessen. Die Bewegung in Abbildung 33 findet im Zeitraum von Frame 1000 bis Frame 2000 statt, wobei der Abstand von 40 cm auf 60 cm erhöht wird. Die darauf folgende Angleichungsphase zwischen dem Ende der Bewegung bis zum Erreichen der optimalen Pose dauert wiederum 500 Frames, also etwa 5 Sekunden. Der maximale Fehler während der Bewegung in Translationsrichting beträgt 9 cm, der entsprechende Rotationsfehler 9 Grad. Die nach der Bewegnung eingenommene Pose liegt mit einem guten Fehler von unter 1 cm und 1 Grad im Bereich des Fehlers der Ausgangspose in Ruhe.

Im zweiten Versuch (Abbildung 34) wird der Abstand von 40 cm auf 80 cm vergrößert. Die Zeit zur Angleichung verdoppelt sich proportional mit dem Abstand auf 1000 Frames. In diesem Fall beträgt die Maximalabweichung in Bewegung 9 cm und 16 Grad, die nach der Bewegung eingenommene Pose weicht mit 1 cm und 4 Grad etwas mehr von der Ausgangslage ab.

Im Mittel wird die Pose im gewählten Ansatz bei einer möglichst optimalen Lichtquelle und gleichbleibendem Abstand mit einer Genauigkeit von 0.5 cm und 1 Grad bestimmt. Die Abhängigkeit der Pose von der Lichtquelle tritt besonders deutlich auf, wenn die Optimierung bei großen Bewegungen in einem lokalen Maximum hängen bleibt. In vielen Fällen kann dieser Zustand durch das Bewegen der Lichtquelle korrigiert werden oder

durch Umschalten in ein größeres Suchfenster. Auch die gemessene Dauer der Angleichung kann durch ein größeres Suchfenster beschleunigt werden. Jedoch ist damit insbesondere bei der Rotation eine größere Ungenauigkeit verbunden.

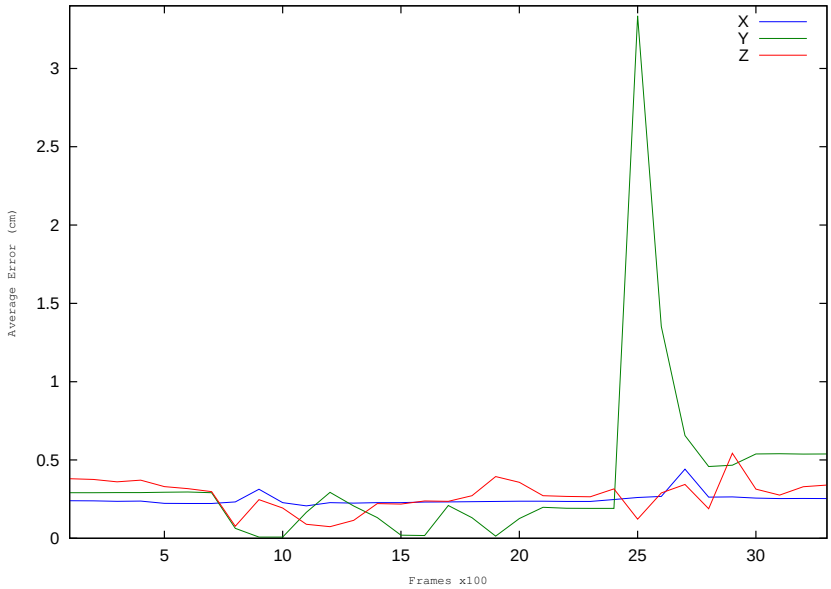

Abbildung 31: Translation

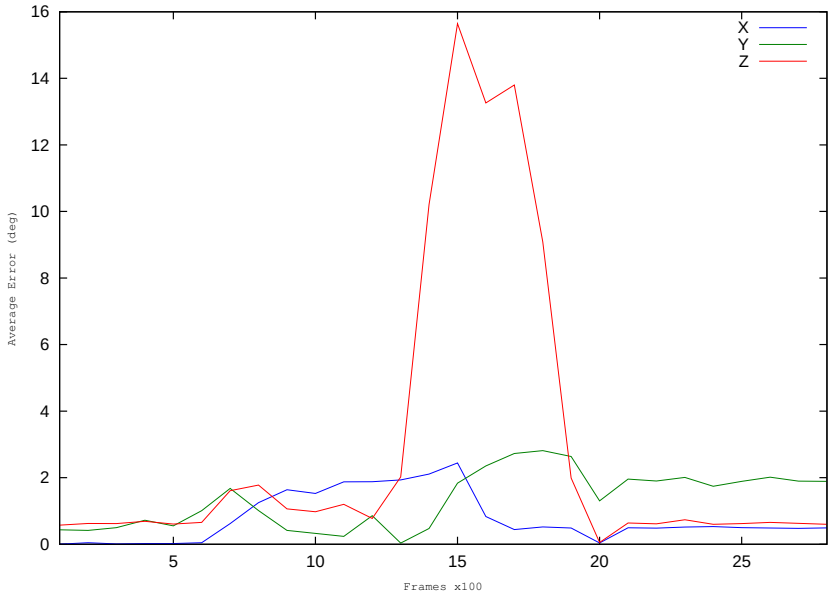

Abbildung 32: Rotation

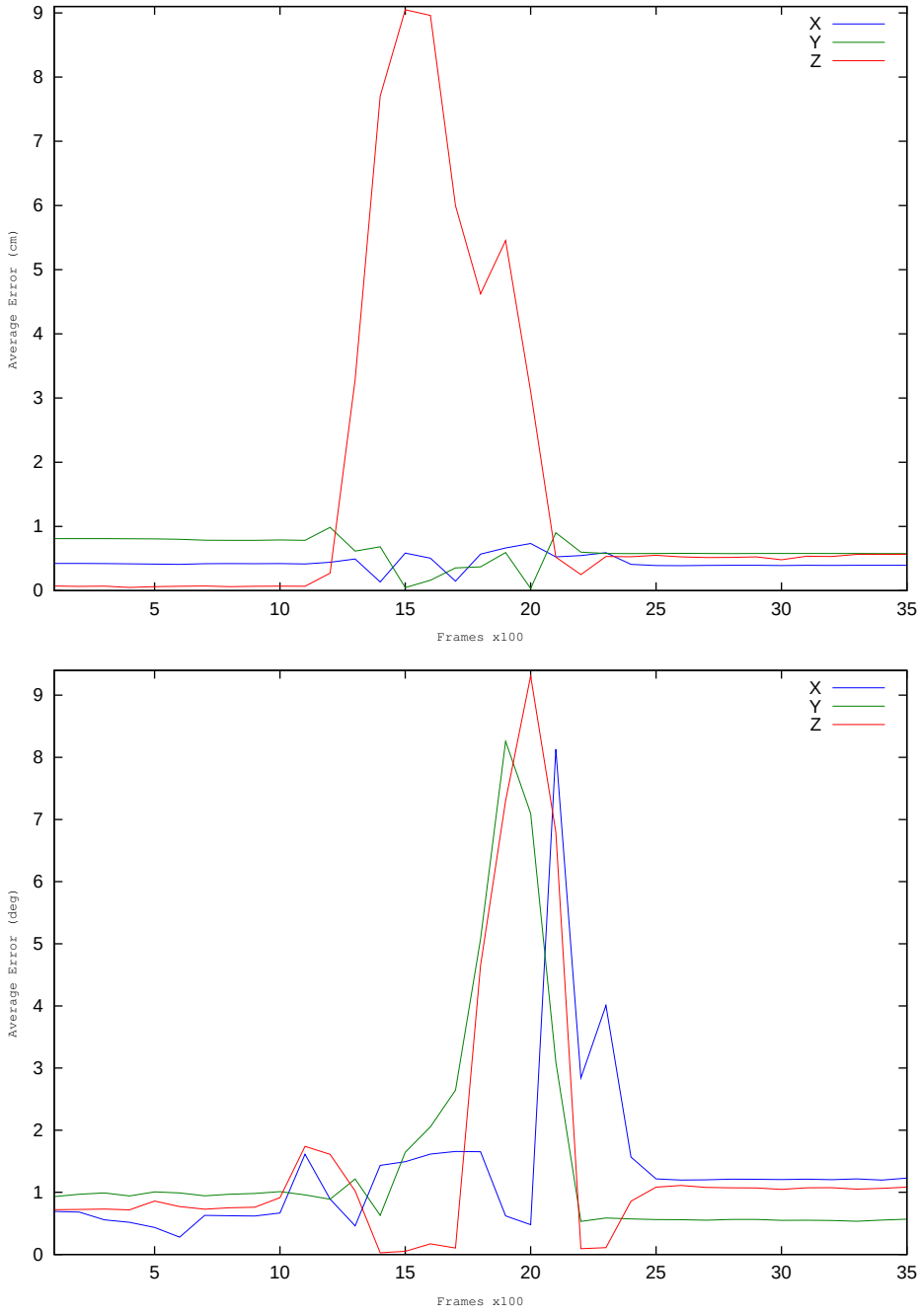

Abbildung 33: Bewegung und Angleichung, Versuch 1

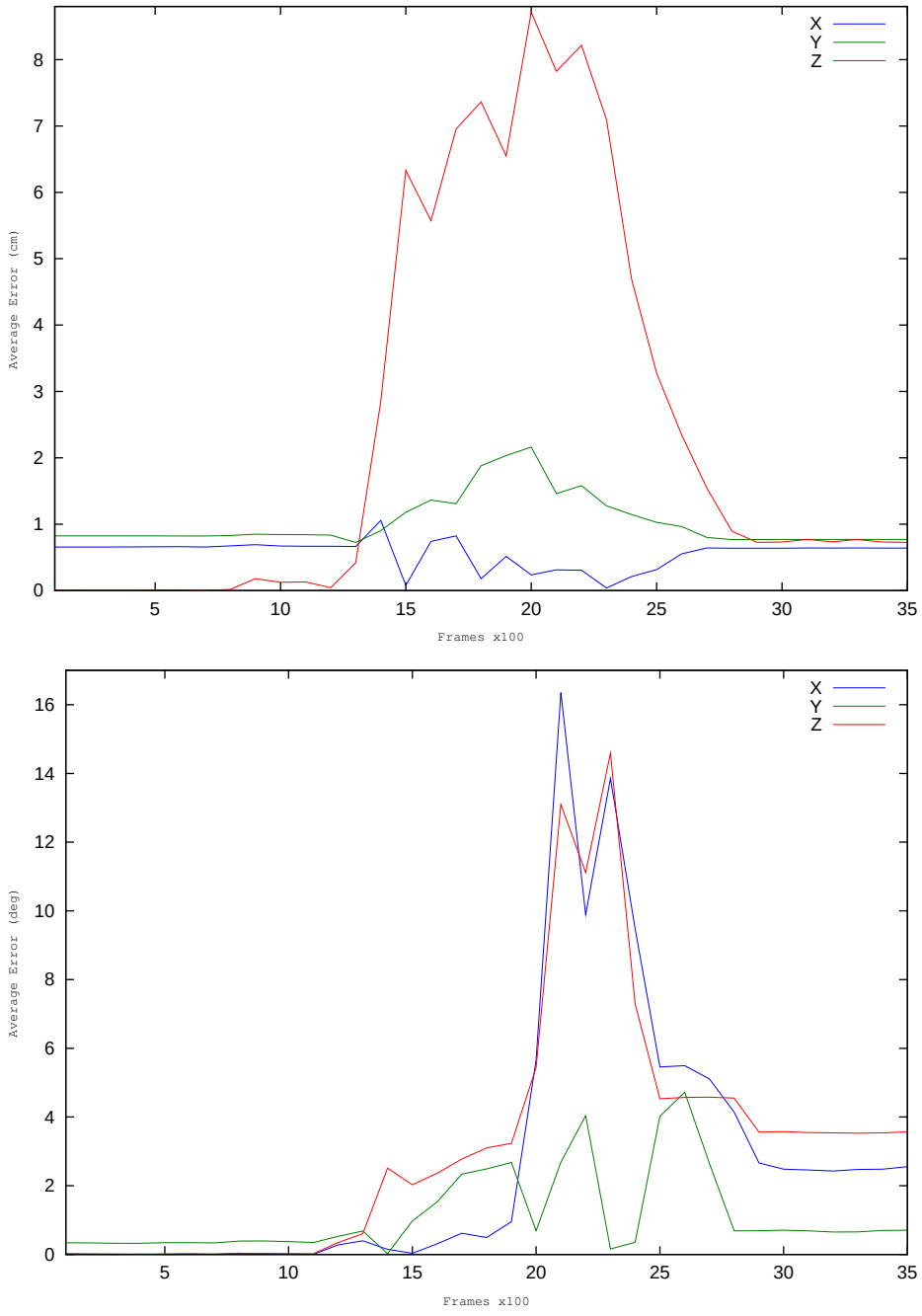

Abbildung 34: Bewegung und Angleichung, Versuch 2

7.2.4 Outdoor

Da die bisherigen Versuche durchweg Box-Tests waren, wird das Tracking abschließend noch auf ein komplexeres Modell mit Texturierung angewendet, um den Ansatz auf Outdoortauglichkeit zu überprüfen. Dazu wird ein Gebäude in einem Video mit dem entsprechenden Gebäudemodell getrackt. Eine genaue Ausmessung des Fehlers ist aufgrund fehlender Referenzkoordinaten nicht möglich. Doch zeigen die Überlagerungen von Kamerabild und Gebäudeobjekt in Abbildung 36 Abweichungen von der Idealpose, die auch optisch deutlich erkennbar sind.

Die gewählte Optimierung eignet sich offensichtlich weniger für Texturen mit feinen Strukturen. Die vielen gleichartigen Elemente im Bild führen bei jeder partiellen Überdeckung, etwa der Kanten, zur Ausbildung vieler gleichähnlicher Bilder und somit zur Bildung lokaler Maxima, die sich störend auswirken. Auch scheint bei texturierten Objekten der Einsatz eines farbbasierten Ähnlichkeitsmaßes sinnvoller zu sein.

Abbildung 35: Kamerabild und Objekt

Abbildung 36: Überlagerung

8 Fazit und Ausblick

Etablierte markerlose Trackingverfahren beruhen auf der Detektion von Merkmalen wie Punkten und Linien, die über eine Bildsequenz verfolgt werden, um aus ihrer Transformation die Pose einer Kamera in Bezug auf das zu trackende Objekt zu bestimmen. In dieser Arbeit wurde gezeigt, dass es auch möglich ist, markerloses Tracking auf Basis der Ähnlichkeitsbestimmung von Bildern zu realisieren. Im Zuge dessen kam die Methode der Analyse durch Synthese zum Einsatz, welche zu einem gegebenen Referenzbild der Kamera eine Anzahl synthetischer Bilder des zu trackenden Objekts aus leicht veränderten Posen erzeugt. Diese von Bildstörungen und Umgebungseinflüssen unabhängigen Vergleichsbilder werden auf Ähnlichkeit zum Referenzbild hin getestet. Die Pose des Bildes mit der größten Ähnlichkeit wird als neue Pose angenommen, aus deren Sicht etwa in Augmented Reality Anwendungen virtuelle Objekte korrekt in das Kamerabild eingeblendet werden können.

Untersucht wurden in diesem Zusammenhang die Tauglichkeit verschiedener Ähnlichkeitsmaße und Bildstilisierungen, mit dem Ziel die Genauigkeit der Pose zu erhöhen. Der Vergleich der Bilder wurde auf Maßen durchgeführt, welche die Intensitätswerte pixelweise auf den Grad der Übereinstimmung testen. Als besonders geeignet für Anwendung im Tracking hat sich dabei die Normalisierte Kreuzkorrelation herausgestellt. Sie liefert bei Vergleich von Kamerabild und gerendetem Bild die stabilsten Werte und stellt eine gute Grundlage für das Optimierungsverfahren dar.

Die Stilisierung von Kamerabild und synthetischem Bild sollte die Vergleichbarkeit erhöhen, indem Bildinformationen hinzugefügt oder reduziert werden, damit die Ähnlichkeitsmaße aussagekräftigere Ergebniswerte liefern. Die Untersuchung verschiedener Darstellungsformen ergab jedoch, dass einfaches diffuses Rendering, also im konkreten Fall die realistischste Darstellung, die besten Ergebnisse beim Vergleich zwischen Kamerabild und synthetischem Bild liefert.

Im Verlauf der Tests wurde weiterhin offensichtlich, dass die Positionierung der virtuellen Lichtquelle einen wesentlich größeren Einfluss auf die Qualität des Trackings hat, als die Bildstilisierung. Im gewählten Ansatz musste die virtuelle Lichtquelle manuell mit der realen in Übereinstimmung gebracht werden, um optimale Ergebnisse zu erzielen. Eine Nachjustierung wurde jedes Mal erforderlich, wenn sich die Kamera relativ zur Lichtquelle bewegte. Eine inkorrekt gesetzte Lichtquelle führte zu größeren Fehlern im Tracking und erhöhte die Wahrscheinlichkeit, in einem lokalen Maximum hängen zu bleiben. Als Verbesserung dieses Umstandes sollte die Verwendung von Lichtquellentracking angedacht werden, um die Qualität der Beleuchtung als Fehlerquelle ausschließen zu können.

Strukturbetonendes Rendern wie etwa Toon Shading führt kaum zu relevanter Verbesserung des Trackings. Ferner erhöht sich bei Einsatz von Stilisierungen die ohnehin schon große Abhängigkeit von der Lichtquelle zusätzlich, sodass eher Nachteile entstehen. Vor diesem Hintergrund wäre die Untersuchung interessant, ob sich als gegenteiliger Ansatz stattdessen ein möglichst photorealistisches Rendering der synthetischen Bilder als geeigneter erweist, das Tracking zu verbessern.

Bei Verwendung komplexer, insbesondere texturierter Modelle, erweist sich die momentane Optimierung der Kamerapose als zu starr bei der Umgehung lokaler Maxima. Texturen können feine Strukturen enthalten, die bei geringer Übereinstimmung bereits hohe Ähnlichkeitswerte erzeugen oder im umgekehrten Fall bei geringer Abweichung die Ähnlichkeitswerte stark reduzieren und so das Auffinden des Maximums erschweren. Eine bessere Optimierungsstrategie sollte daher zur Vermeidung lokaler Maxima verschiedene Ansätze für Grob- und Feintracking kombinieren.

Problematisch ist bei der aktuellen Optimierung auch das Finden des richtigen Suchfensters, in dem die neuen Kameraposen erzeugt werden. Der Schwellwert als Kriterium zum Umschalten zwischen verschiedenen Suchfenstergrößen ist fest gewählt und abhängig von der aktuell gemessenen Ähnlichkeit. Diese sinkt jedoch bei beliebigem Bildhintergrund und Fremdobjekten im Bild, was zur Auswahl falscher Suchfenstergrößen führt und damit die Qualität des Trackings verschlechtert oder es ganz unmöglich macht. Aufgrund dessen wurden die Versuche mit einheitlichem Hintergrund durchgeführt. Hier gilt es, eine passende Methode zur Figur-Grund-Trennung zu finden, damit auch zuverlässiges Outdoor-Tracking möglich wird.

Insgesamt wurden mit diesem Ansatz, die Präzision des Trackings betreffend, gute Ergebnisse erzielt. Die Analyse durch Synthese erwies sich im Zusammenhang mit globalen Ähnlichkeitsvergleichen von Bildern als interessante Möglichkeit, markerloses Tracking zu realisieren. Wenngleich die Stilisierung von Bildern nicht den erhofften positiven Effekt auf die Qualität des Trackings hatte, wurde eine Grundlage geschaffen, die unter entsprechenden Verbesserungen ausbaufähig ist. Zu untersuchen wäre weiterhin, ob etwa nicht-intensitätsbasierte Ähnlichkeitsmaße Vorteile bei der Genauigkeit oder der Geschwindigkeit bringen und ob ein kombinierter Ansatz mit merkmalsbasierten Elementen Erfolg versprechen könnte. Verbesserungen sind sicherlich für das Lichtquellenhandling und eine alternative Art der Optimierung zu erwarten. Für eine Steigerung der Geschwindigkeit wäre es nicht zuletzt sinnvoll, die Berechnung der Ähnlichkeit auf die GPU zu verlagern.

A Anhang

A.1 Shader-Klasse

A.1.1 shader.h

```
#ifndef SHADER_H
#define SHADER_H

#include <GL/glew.h>
#include <GL/glut.h>
#include <iostream>

class Shader
{
  public:

   Shader(char*, char*);
   void init();
   void enableShaders();
   void disableShaders();
   void setUniform1i(GLchar* name, GLint value);
   void setUniform1f(GLchar* name, GLfloat value);
   void printProgramInfoLog(GLuint obj);

  protected:

  private:

   GLuint vertex, fragment, program;
};

#endif
```

A.1.2 shader.cpp

```
#include "shader.h"
using namespace std;

// Laden und Kompilieren der Shader
Shader::Shader(char *vert, char *frag)
{
  char *pVertex = NULL, *pFragment = NULL;
  vertex   = glCreateShader(GL_VERTEX_SHADER);
  fragment = glCreateShader(GL_FRAGMENT_SHADER);
```

```
  // Sei readFile eine beliebige Funktion
  // zum Auslesen von Textdateien
  pVertex   = readFile(vert);
  pFragment = readFile(frag);

  const char * vv = pVertex;
  const char * ff = pFragment;

  glShaderSource(vertex,   1, &vv, NULL);
  glShaderSource(fragment, 1, &ff, NULL);

  free(pVertex);
  free(pFragment);

  glCompileShader(vertex);
  glCompileShader(fragment);

  program = glCreateProgram();
  glAttachShader(program, vertex);
  glAttachShader(program, fragment);
  glLinkProgram(program);

  printProgramInfoLog(program);
}

// Initialisierung
void Shader::init()
{
  glewInit();

  if (glewIsSupported("GL_VERSION_2_0"))
    printf("\nOpenGL 2.0 supported\n");
  else{
    printf("\nOpenGL 2.0 not supported\n");
    exit(1);
  }

  printf("OpenGL Vendor: %s\n",    glGetString(GL_VENDOR));
  printf("OpenGL Renderer: %s\n",  glGetString(GL_RENDERER));
  printf("OpenGL Version: %s\n\n", glGetString(GL_VERSION));
  printf("GLSL Version: %s\n\n",
         glGetString(GL_SHADING_LANGUAGE_VERSION));
}
```

```
// Statusmeldungen, Ausgabe von Fehlern
void Shader::printProgramInfoLog(GLuint obj)
{
  int infologLength = 0;
  int charsWritten  = 0;
  char *infoLog;

  glGetProgramiv(obj, GL_INFO_LOG_LENGTH,&infologLength);

    if (infologLength > 0)
    {
        infoLog = (char *)malloc(infologLength);
        glGetProgramInfoLog(obj, infologLength,
                            &charsWritten, infoLog);
        printf("%s\n",infoLog);
        free(infoLog);
    }
}

// Aktivierung des Shaders
void Shader::enableShaders()
{
  glUseProgram(program);
}

// Deaktivierung des Shaders
void Shader::disableShaders()
{
  glUseProgram(0);
}

// Übergabe von Integern an den Shader
void Shader::setUniform1i(GLchar *name, GLint value)
{
  glUniform1i(glGetUniformLocation(program, name), value);
}

// Übergabe von Float an den Shader
void Shader::setUniform1f(GLchar *name, GLfloat value)
{
  glUniform1f(glGetUniformLocation(program, name), value);
}
```

A.2 Geometrieshader

A.2.1 Vertexshader Toon-Shading

```
//Variablen zur Übergabe an den Fragmentshader
varying vec3 normal, light;

void main()
{
  // Transformation des Vertex und der Normale
  vec3 vertex = vec3(gl_ModelViewMatrix * gl_Vertex);
  normal = gl_NormalMatrix * gl_Normal;

  // Berechnung des Vektors zur Lichtquelle
  light = vec3(gl_LightSource[0].position) - vertex;

  gl_Position = ftransform();
}
```

A.2.2 Fragmentshader Toon-Shading

```
// Variablen vom Vertexshader
varying vec3 normal, light;

void main()
{
  float intensity;
  vec4 color;

  // Lichtintensität durch Skalarprodukt
  // von Normale und Lichtvektor
  intensity = max(dot(normalize(light),normalize(normal)),0.0);

  // Bestimmung der Farbe nach Lichtintensität
  if (intensity > 0.98)
    color = vec4(0.8,0.8,0.8,1.0);
  else if (intensity > 0.5)
    color = vec4(0.4,0.4,0.8,1.0);
  else if (intensity > 0.25)
    color = vec4(0.2,0.2,0.4,1.0);
  else
    color = vec4(0.0,0.0,0.0,1.0);

  gl_FragColor = color;
}
```

A.2.3 Vertexshader Gooch-Shading

```
// Variablen zur Übergabe an den Fragmentshader
varying float dotProduct;
varying vec3  reflection, viewVector;

void main()
{
  // Transformation des Vertex und der Normale
  vec3 vertex = vec3(gl_ModelViewMatrix * gl_Vertex);
  vec3 normal = normalize(gl_NormalMatrix * gl_Normal);

  // Berechnung des Skalarprodukts und der Reflektion
  vec3 lightPos = vec3(gl_LightSource[0].position);
  vec3 lightVec = normalize(lightPos - vertex);
  reflection    = normalize(reflect(-lightVec, normal));
  viewVector    = normalize(-vertex);
  dotProduct    = (dot(lightVec, normal) + 1.0) * 0.5;

  gl_Position   = ftransform();
}
```

A.2.4 Fragmentshader Gooch-Shading

```
vec3  SurfaceColor = vec3(0.75, 0.75, 0.75);
vec3  WarmColor = vec3(0.6, 0.6, 0.0);
vec3  CoolColor = vec3(0.0, 0.0, 0.6);
float DiffuseWarm = 0.45;
float DiffuseCool = 0.45;

// Variablen vom Vertexshader
varying float dotProduct;
varying vec3  reflection, viewVector;

void main()
{
  // Kombination von warmer, kalter und Oberflächenfarbe
  vec3 cool = min(CoolColor + DiffuseCool
                      * SurfaceColor,1.0);
  vec3 warm = min(WarmColor + DiffuseWarm
                      * SurfaceColor,1.0);

  // Interpolation durch das Skalarprodukt
  vec3 final = mix(cool, warm, dotProduct);
```

```
  // Berechnung des spekularen Highlights
  vec3 nreflection = normalize(reflection);
  vec3 nviewVector = normalize(viewVector);
  float specular   = max(dot(nreflection, nviewVector), 0.0);
  specular         = pow(specular, 32.0);

  gl_FragColor     = vec4(min(final + specular, 1.0), 1.0);
}
```

A.3 Textursha der

A.3.1 Vertexshader für Texturbilder

Vertexshader für die Stilisierung von Texturbildern. Lässt die Standardfunktion der Grafikpipeline unverändert und gibt nur die notwendigen Texturkoordinaten an den Fragmentshader weiter.

```
void main()
{
  gl_Position    = ftransform();
  gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

A.3.2 Fragmentshader Mittelwertfilter

```
// Variable für die Übergabe der Textur
uniform sampler2D image;

// Pixeloffset über Breite und Höhe der Textur
float xOff = 1.0/500.0;
float yOff = 1.0/375.0;

void main()
{
  // Auslesen einer 3x3 Umgebung
  vec2 coord = gl_TexCoord[0].xy;
  vec4 c  = texture2D(image, coord);
  vec4 bl = texture2D(image, coord + vec2(-xOff, -yOff));
  vec4 l  = texture2D(image, coord + vec2(-xOff,   0.0));
  vec4 tl = texture2D(image, coord + vec2(-xOff,  yOff));
  vec4 t  = texture2D(image, coord + vec2(  0.0,  yOff));
  vec4 tr = texture2D(image, coord + vec2( xOff,  yOff));
  vec4 r  = texture2D(image, coord + vec2( xOff,   0.0));
  vec4 br = texture2D(image, coord + vec2( xOff, -yOff));
  vec4 b  = texture2D(image, coord + vec2(  0.0, -yOff));

  // Mittelwertfilter
  gl_FragColor = (c+b+t+l+r+tl+tr+bl+br) / 9.0;
}
```

A.3.3 Fragmentshader Sobelfilter

```
// Variable für die Übergabe der Textur
uniform sampler2D image;

// Pixeloffset über Breite und Höhe der Textur
float xOff = 1.0/500.0;
float yOff = 1.0/375.0;

vec4 gradientX = vec4(0.0,0.0,0.0,1.0);
vec4 gradientY = vec4(0.0,0.0,0.0,1.0);
vec4 magnitude = vec4(0.0,0.0,0.0,1.0);
float average = 0.0;

void main()
{
  // Auslesen einer 3x3 Umgebung
  vec2 coord = gl_TexCoord[0].xy;
  vec4 c  = texture2D(image, coord);
  vec4 bl = texture2D(image, coord + vec2(-xOff, -yOff));
  vec4 l  = texture2D(image, coord + vec2(-xOff,   0.0));
  vec4 tl = texture2D(image, coord + vec2(-xOff,  yOff));
  vec4 t  = texture2D(image, coord + vec2(  0.0,  yOff));
  vec4 tr = texture2D(image, coord + vec2( xOff,  yOff));
  vec4 r  = texture2D(image, coord + vec2( xOff,   0.0));
  vec4 br = texture2D(image, coord + vec2( xOff,  yOff));
  vec4 b  = texture2D(image, coord + vec2(  0.0, -yOff));

  // Sobel-Operation
  gradientX = 2.0 * (l - r) + tl + bl - tr - br;
  gradientY = 2.0 * (t - b) + tl + tr - bl - br;
  magnitude = abs(gradientX) + abs(gradientY);

  // Grauwert als Schwellwert für Kantenpixel
  average = (magnitude.r + magnitude.g + magnitude.b)/3.0;

  if ( average >= 0.2 )
  {
    // Setze weißes Pixel für Kante
    gl_FragColor = vec4(1.0,1.0,1.0,1.0);
  }else{
    gl_FragColor = vec4(0.0,0.0,0.0,1.0);
  }
}
```

A.3.4 Fragmentshader Gradientenwinkel

```
// Variable für die Übergabe der Textur
uniform sampler2D image;
// Pixeloffset über Breite und Höhe der Textur
float xOff = 1.0/500.0;
float yOff = 1.0/375.0;

vec4 gradientX = vec4(0.0,0.0,0.0,1.0);
vec4 gradientY = vec4(0.0,0.0,0.0,1.0);
vec4 color = vec4(0.0,0.0,0.0,0.0);
float greyX, greyY, magnitude, angle, tangent;

void main()
{
  [... Auslesen der Textur wie Sobel ...]
  gradientX = 2.0 * (l - r) + tl + bl - tr - br;
  gradientY = 2.0 * (t - b) + tl + tr - bl - br;
  greyX = (gradientX.r + gradientX.g + gradientX.b)/3.0;
  greyY = (gradientY.r + gradientY.g + gradientY.b)/3.0;
  magnitude = abs(greyX) + abs(greyY);

  // Berechnung des Gradientenwinkels
  tangent = (atan(greyY,greyX));
  angle = degrees(tangent);

  // Kodierung des Winkels durch Farbkanten
  // Anwendung kann Winkel über Alphawert auslesen
  if ( magnitude >= 0.2 )
  {
    if (angle < -135.0) color = vec4(1.0,0.0,0.0,0.1); else
    if (angle <  -90.0) color = vec4(0.0,1.0,0.0,0.2); else
    if (angle <  -45.0) color = vec4(0.0,0.0,1.0,0.3); else
    if (angle <    0.0) color = vec4(1.0,1.0,0.0,0.4); else
    if (angle <   45.0) color = vec4(1.0,0.0,0.0,0.5); else
    if (angle <   90.0) color = vec4(0.0,1.0,0.0,0.6); else
    if (angle <  135.0) color = vec4(0.0,0.0,1.0,0.7); else
                        color = vec4(1.0,1.0,0.0,0.8);

    gl_FragColor = color;
  }else{
    gl_FragColor = vec4(0.0,0.0,0.0,1.0);
  }
}
```

A ANHANG

Literatur

[Aar97] Aarts E., Lenstra J. K.: *Local Search in Combinatorial Optimization*. Wiley & Sons, New York, 1997

[Ach08] Achilles S.: *Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis von Featuredetektoren*. Diplomarbeit, Universität Koblenz-Landau, Koblenz, 2008.

[Bar08] Barthel K.U. et al.: *Improved Image Retrieval Using Visual Sorting and Semi-Automatic Semantic Categorization of Images*. Zentrum für Mensch-Maschine-Kommunikation, FHTW Berlin, 2008. `http://mmk.f4.fhtw-berlin.de/Projekte/ImageSorter` (Zugriff 31.03.2008)

[Bel61] Bell C.G. et al.: *Reduction of Speech Spectra by Analysis-by-Synthesis Techniques*. The Journal of the Acoustical Society of America, vol 33(12), pp 1725-1736, 1961.

[Bro92] Brown L.G.: *A survey of image registration techniques*. ACM Computing Surveys (CSUR), vol 24(4), pp 326-376, 1992.

[Cox95] Cox G.S.: *Template Matching and Measures of Match in Image Processing*. Department of Electrical Engineering, University of Cape Town, 1995.

[Den07] Dennhardt M.: *Kamerapositionsbestimmung über Analyse durch Synthese*. Diplomarbeit, Universität Koblenz-Landau, Koblenz, 2007.

[Ewe06] Ewering D.: *Modellbasiertes Tracking mittels Linien- und Punktkorrelationen*. Diplomarbeit, Universität Koblenz-Landau, Koblenz, 2006.

[Fer06] Fernandes A.R.: *GLSL Tutorial*. 2006 `http://www.lighthouse3d.com/opengl/glsl/` (Zugriff 05.06.2008)

[Goo98] Gooch A., Gooch B. et al.: *A Non-Photorealistic Lighting Model for Automatic Technical Illustration*. Computer Graphics (SIGGRAPH '98 Proceedings), pp 447-452, 1998.

[Intel] Intel Corporation: Open Source Computer Vision Library, OpenCV 1.0 `http://sourceforge.net/projects/opencvlibrary/` `http://www.intel.com/technology/computing/opencv/index.htm` (Zugriff 27.01.2008)

[ArLib] Kato H., Billinghurst M.: ARToolKit 2.72.1. Human Interface Technology Laboratory (HITLab), University of Washington; HIT Lab NZ, University of Canterbury, New Zealand. `http://www.hitl.washington.edu/artoolkit/` (Zugriff 27.01.2008)

[Lie89] Liedtke C., Ender M.: *Wissensbasierte Bildverarbeitung*. Springer Verlag, 1989.

[Moe99] Moeslund T. B.: *The Analysis-by-Synthesis Approach in Human Motion Capture: A Review*. The 8th Danish conference on pattern recognition and image analysis. Copenhagen University, Copenhagen, 1999.

[Nis04] Nischwitz A., Haberäcker P.: *Masterkurs Computergrafik und Bildverarbeitung*. Friedr. Vieweg & Sohn Verlag / GWV Fachverlage GmbH, Wiesbaden, 2004.

[Pau01] Paulus D.: *Aktives Bildverstehen*. Der Andere Verlag, Osnabrück, 2001.

[Ros04] Rost R. J.: *OpenGL Shading Language*. Addison Wesley, 2004.

[Rub01] Rubner Y. et al.: *Empirical Evaluation of Dissimilarity Measures for Color and Texture*. Computer Vision and Image Understanding, vol 84(1), pp 25-43, 2001.

[San99] Santini S., Jain R.: *Similarity Measures*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 21(9), 1999.

[Swe02] Schlechtweg S., Strothotte T.: *Non-Photorealistic Computer Graphics - Modeling, Rendering and Animation*. Morgan Kaufmann Publishers, 2002.

[Smi06] Schmitt, I.: *Ähnlichkeitssuche in Multimedia-Datenbanken: Retrieval, Suchalgorithmen und Anfragebehandlung*. Oldenbourg Verlag, München, 2006

[Sol04] Scholz O. R.: *Bild, Darstellung, Zeichen*. Seminar Klostermann, Vittorio Klostermann GmbH, Frankfurt, 2. Auflage 2004.

[Sch07] Schumann M.: *Augmented Reality und Non-Photorealistic Rendering*. Studienarbeit, Universität Koblenz-Landau, Koblenz, 2007.

[Str02] Stricker D.: *Computer-Vision-basierte Tracking- und Kalibrierungsverfahren für Augmented Reality*. Dissertation, Fachbereich Informatik, Technische Universität Darmstadt, 2002.

[Ton94] Tönjes R. et al.: *Analyse durch Synthese Modellierung von 3D-Objekten in Stereobildfolgen*. 1. Workshop visual computing, Darmstadt, 1994.

[QtLib] Trolltech Company: Qt Framework, Version Qt 4.3.3. `http://www.trolltech.com` (Zugriff 02.02.2008)

[Wei03] Wein W.: *Intensity Based Rigid 2D-3D Registration Algorithms for Radiation Therapy*. Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2003

[Wil78] Williams L.: *Casting Curved Shadows on Curved Surfaces*. ACM SIGGRAPH Computer Graphics, vol 12(3), pp 270-274, 1978.

[Wue07] Wuest H., Stricker D.: *Tracking of industrial objects by using CAD models*. Journal of Virtual Reality and Broadcasting, vol 4(1), 2007.

[Zit03] Zitová B., Flusser J.: *Image registration methods: a survey*. Image and Vision Computing, vol 21(11), pp 977-1000, 2003.