UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Realisierung einer eigenständigen Trackingbibliothek auf der Basis des ARToolkit und des ARToolkitPlus

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von

Martina Brümmer

Betreuer: Prof. Dr.-Ing. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im April 2007

Erklärung

	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

1 Einleitung

1.1 Motivation

Augmented Reality (AR) ist ein Teilbereich der Informatik, der sich mit der Darstellung von virtuellen Objekten in der Überlagerung mit realen Bildern beschäftigt. Wie der Name schon sagt, handelt es sich um eine Erweiterung der Realität, im Gegensatz zur *Virtual Reality*, die eine rein virtuell erstellte Welt offenbart. Mit dem Bereich der *Augmented Reality* ist ein umfangreiches, aber bisher noch nicht voll genutztes Potenzial der Informatik in der Entwicklung, das in einigen Bereichen, wie Medizin, Design, Wartung und Montage schon einige interessante Anwendungen und Ansätze geschaffen hat.
Die Idee, die dieser Arbeit zugrunde liegt, ist es, die *Augmented Reality* auch in anderen Bereichen voranzutreiben. In der Filmindustrie beispielsweise behilft man sich schon seit langem mit sowohl virtuellen als auch realen Methoden (computergestützten Visualisierungen, Miniatur-Kulissen), um eine Vorvisualisierung der Dreharbeiten zu erhalten, welche zur Planung des Arbeitsablaufs verwendet werden können. Die Idee liegt hierbei darin, dass durch ein Werkzeug, welches sich der *Augmented Reality* bedient, zum Beispiel Belichtungsverhältnisse bereits im Voraus ausgetestet werden könnten, oder der Kameramann seine Einstellungen proben kann. So können hierfür mitunter virtuelle Objekte in eine Miniaturszene eingeblendet werden, mit denen die realen Voraussetzungen des Drehorts nachgeahmt werden. Um diese Vorstellung von einem Werkzeug für die Filmindustrie zu ermöglichen, wird ein gutes und stabiles Tracking benötigt, das die nötigen Eigenschaften zur Verfügung stellt. Um ein solches Trackingsystem geht es in dieser Studienarbeit.

1.2 Projektidee

Diese Arbeit basiert auf einer Idee, die in Zusammenhang mit der Ludwigsburger Filmakademie[1] entstand. Die dortigen Studenten arbeiten täglich an verschiedensten Filmprojekten und sind daher immer auf der Suche nach Möglichkeiten, wie sie ihren Arbeitsablauf erleichtern können. So entstand auch die Idee, für die Vorvisualisierung von Filmszenen auf die *Augmented Reality* zurückzugreifen. In Zusammenarbeit mit der Universität Koblenz wurde ein Plan erarbeitet, ein solches Projekt umzusetzen. Dabei fiel die Entwicklung eines passenden Trackingsystems der Universität Koblenz zu. Darauf aufbauend entstehen parallel zwei Previsualisierungstools, eines an der Ludwigsburger Filmakademie basierend auf dem Programm Touch[2] und eines an der Universität Koblenz basierend auf dem Programm Maya[3].

[1] http://www.filmakademie.de
[2] http.//www.derivativeinc.com/tools/tools.asp
[3] http.//www.alias.com

1.3 Ziel

Ziel dieser Arbeit ist es, ein markerbasiertes Objekttracking zu realisieren, das insbesondere auf das oben genannte Previsualisierungswerkzeug zugeschnitten ist. Dabei soll kein neues Trackingsystem entwickelt werden, sondern auf vorhandene Systeme, wie zum Beispiel das *ARToolkit* oder das *ARToolkitPlus* aufgebaut werden. Der erste Teil der Arbeit besteht darin, ausgewählte vorhandene markerbasierte Trackingsysteme zu untersuchen und das für die Aufgabe Sinnvollste auszuwählen.
Weiterhin sollten für das Projekt gewisse Anforderungen erfüllt sein. Es soll ein möglichst robustes und stabiles Tracking vorhanden sein, welches insbesondere auch Schwankungen in den Beleuchtungssituationen ausgleichen muss. Gerade in Filmszenen sind die unterschiedlichsten Helligkeitssituationen vorhanden, und nicht immer eine perfekt ausgeleuchtete Umgebung. Auch soll es möglich sein, mehrere Marker eindeutig zu detektieren und ihnen anhand ihrer ID eindeutig Objekte zuordnen zu können. Als ein Hauptziel wurde definiert, dass alle Marker in einem Weltkoordinatensystem platziert sind, welches sich an einem Weltmarker orientiert. Alle weiteren Marker müssen in Beziehung zu diesem Weltmarker ihre Position und Orientierung angeben. Diese Daten sollen möglichst einfach abzurufen sein. Es ist vorgesehen das gesamte neue System als kompakte, leicht zu installierende und ebenso leicht zu bedienende Einheit zu erstellen. Als letzte Anforderung wurde definiert, dass das Gesamtsystem auf einem Notebook mit dem Betriebssystem Windows ausführbar sein muss.

1.4 Aufbau der Arbeit

Im ersten Teil dieser Ausarbeitung werden die allgemeinen Grundlagen, die für diese Arbeit notwendig sind, vorgestellt. Dabei wird kurz auf das Thema Augmented Reality und Marker Tracking im Allgemeinen eingegangen. Ausführlicher wird hingegen die Funktionsweise und die Struktur zweier Trackingsysteme, das *ARToolkit* und das *ARToolkitPlus*, beleuchtet. Alternative Systeme werden nur kurz angeschnitten.
Der Abschluss des Kapitels 2 erläutert und begründet die getroffene Wahl des Trackingssystems. Im Kapitel 3 geht es um die Implementierung des neuen realisierten Systems. Dabei wird im ersten Teil die Videoanbindung näher betrachtet, und im zweiten Teil das Tracking für die oben genannten Anforderungen erweitert. Am Schluss diese Arbeit wird das Gesamtergebnis vorgestellt und ein Ausblick gegeben.

2 Grundlagen

2.1 Augmented Reality und Tracking

Augmented Reality ist ein Teilbereich der Computergrafik, der sich mit der Darstellung von virtuellen Objekten in realen Bildern beschäftigt. Dabei geht es vor allem darum, dem Benutzer zusätzliche Informationen in Überlagerung mit einer realen Szene bereitzustellen. Diese Informationen können virtuelle Objekte oder auch Text sein, die in einem aufgenommenen Kamerabild an der richtigen Stelle eingeblendet werden, so dass der Benutzer durch diese zusätzlichen Angaben unterstützt wird.[6] So kann zum Beispiel ein Mechaniker sich verdeckte Teile, oder zusätzliche Hilfsinformationen virtuell anzeigen lassen, siehe Abbildung 1.
Es ist also kein Wunder, dass *Augmented Reality* bereits in vielen Bereichen Anwendung findet, wie zum Beispiel in der Medizin, in der Unterhaltungsindustrie (AR-Spiele), für mobile Informationssysteme, im Design oder in der Wartungs- und Montageunterstützung.[6] Dennoch ist ihr Potenzial noch lange nicht ausgereizt.

Abbildung 1: Wartung mit Hilfe von AR[7]

Ein wichtiger Punkt bei der Verwendung von Systemen im Bereich der *Augmented Reality* ist das Tracking. Tracking bedeutet im weitesten Sinn das Finden und Verfolgen von bewegten Objekten. Hierfür gibt es verschiedenste Methoden, zum Beispiel mechanisches Tracking, Ultraschall, Infrarot, GPS oder optisches Tracking, um nur einige zu nennen. Viele bekannte AR-Systeme, wie auch das *ARToolkit* benutzen das optische Tracking, bei dem noch zwischen zwei verschiedenen Arten unterschieden wird: *Out-side-in* und *In-side-Out* [6]. Beim *Out-side-in* Verfahren werden an dem Objekt passive oder aktive Marker angebracht, welche durch mehrere feste Kameras in der Umgebung aufgenommen werden. Durch die geometrische Anordnung der Marker wird die Position und Orientierung des Objektes bestimmt. In vielen AR-Systemen findet dagegen das *In-Side-Out* Verfahren Verwendung.

Dieses Verfahren wird auch markerbasiertes Tracking genannt, da hierbei in der Umgebung verteilte Marker durch eine Kamera verfolgt werden. Mit Hilfe dieser Marker ist es nach Berechnung ihrer Position und Orientierung möglich, die Überdeckung des virtuellen mit dem realen Bild herzustellen. Nachteil dieser Methode ist es jedoch, das die Marker selbst immer sichtbar sind, auch für unbeteiligte Personen. Es ist aber eine gelungene Alternative zu kostspieligeren Varianten des Trackings, da die Marker einfach und schnell ausgedruckt werden können. Außerdem ist keinerlei Verkabelung notwendig, was das Ganze sehr mobil macht.

2.2 Das ARToolkit

Bei der Entwicklung einer *Augmented Reality* Anwendung stößt man bei der Planung nahezu unvermeidlich auf das *ARToolkit*, da dieses für nicht kommerzielle Zwecke ein frei benutzbares markerbasiertes Trackingverfahren zur Verfügung stellt. Die hierfür benötigten Funktionen finden sich alle in einer Bibliothek wieder, gegen die einfach gelinkt werden kann. Damit aber einfache AR-Applikationen möglichst leicht realisiert werden können, haben die Entwickler das Toolkit als eine sogenannte „Out-of-the-Box“ Lösung konzipiert. Das bedeutet, das der Benutzer einfach eine Kamera anschließen und das Projekt auf dem jeweiligen Betriebssystem kompilieren kann. Von diesem Standpunkt aus ist es schon möglich eine kleine AR-Anwendung („simple test“) zu starten, wo ein ausgedruckter Marker getrackt und ein kleiner virtueller Würfel darauf gezeichnet wird (Dieses Programm ist dann beliebig erweiterbar). Das *ARToolkit* stellt also außer den Tracking-Routinen, auch noch Funktionen für die Videoanbindung und für die Darstellung der Szenen und der Objekte zur Verfügung.

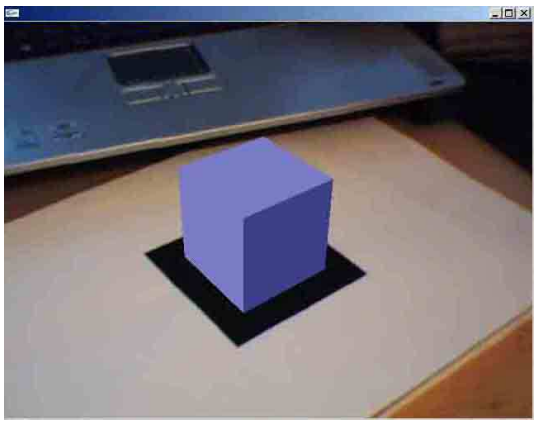

Abbildung 2: Programm simpleTest des *ARToolkit*

Ebenso ist es plattformunabhängig, dass heißt mit den richtigen Treibern läuft es auf den Betriebsystemen Windows, MacOSX und Linux [3]. Da

schon in den Anforderungen definiert worden ist, dass das Betriebssystem Windows benutzt werden soll, ist das *ARToolkit* auch nur hiermit getestet worden. Ebenso wurde die Videoanbindung für das Gesamtsystem auf dieses Betriebssystem ausgelegt. Die Funktionalität für MacOSX und Linux kann aber jederzeit hinzugefügt werden.

2.2.1 Funktionsweise

Um die Methodik, welche hinter dem *ARToolkit* steckt zu verstehen, ist es sinnvoll sich die Funktionsweise, also eine Übersicht dessen, wie das System überhaupt arbeitet, vor Augen zu führen. Diese ist im Grunde ganz einfach und lässt sich in ein paar wenige, aber wichtige Schritte unterteilen, die in einer Art Pipeline durchlaufen werden(Abb. 4).

Im ersten Schritt wird von einer angeschlossenen Kamera ein gerade aufgenommenes Videobild geholt und zur Verarbeitung bereit gestellt. Daraufhin wird mittels der Algorithmen des *ARToolkits* [3] nach Markern in diesem Bild gesucht, wobei zunächst nur die quadratischen schwarzen Rahmen der Marker interessant sind. Wenn ein solcher Rahmen gefunden wurde, wird durch mathematische Berechnungsroutinen die Position des Markers zur Kamera bestimmt [3], siehe Abbildung. 3. Näheres zu den Koordinatensystemen untereinander im Abschnitt 2.5

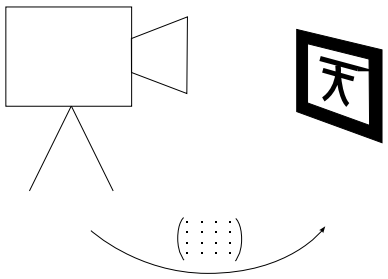

Abbildung 3: Transfomationsmatrix des *ARToolkit*

Im nächsten Schritt wird das Symbol im Inneren des quadratischen Rahmens identifiziert. Dabei wird das Muster (Pattern) mit bereits zuvor in das Programm Geladenen verglichen, und das mit der höchsten Übereinstimmung ausgewählt. Ab diesem Zeitpunkt ist also die Information über ID und Position des Markers bekannt. Mit besagten Angaben ist es daraufhin möglich, ein beliebiges virtuelles Objekt an diese Marker-ID zu binden, und dieses an der Stelle, wo der Marker im realen Bild liegt, zu überblenden. Dies geschieht mit Hilfe der Transformationsmatrix, die aus der Positionsbestimmung des Markers zur Kamera ermittelt wurde. Das Objekt wird genau so positioniert, dass es Orientierung und Standpunkt des Markers übernimmt. Als letztes

wird das virtuelle Objekt in das Videobild gerendert und das augmentierte Bild dargestellt. Pro Frame werden die Schritte dann wiederholt.[3]

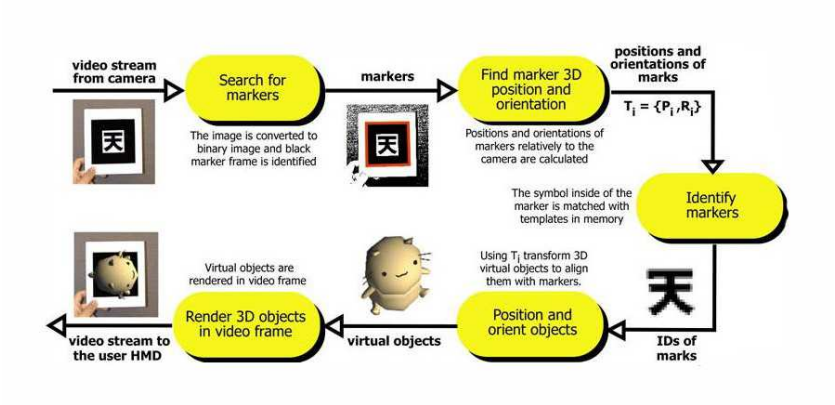

Abbildung 4: Funktionsweise des *ARToolkit* [3]

Diese Beschreibung der Funktionsweise ist nur ein grober Überblick über die Leistung und das Vorgehen des *ARToolkit*. Bis auf die Videoanbindung wird auf die genauen Funktionen dieser Schritte nicht näher eingegangen, da für das Tracking im Gesamtsystem das *ARToolkitPlus* gebraucht wird. Der Ablauf ist aber in beiden Systemen identisch.

2.2.2 Struktur

Das *ARToolkit* ist eine Funktionsbibliothek, die aus verschiedenen Teilen besteht. Das System benutzt dabei OpenGL für die Darstellung und GLUT für die Fensterverwaltung. Ebenso setzt es auf die betriebssystemsspezifische Standard-API und VideoLibrary auf. Diese vier Teile regeln dann den Austausch mit den rechnerinternen Video- und Grafiktreibern.

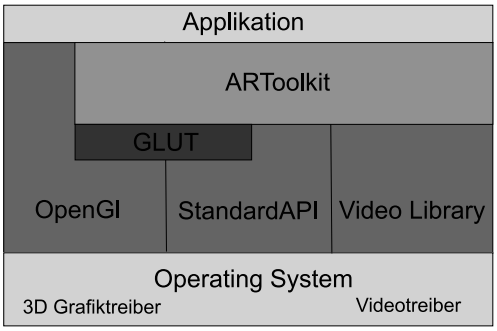

Abbildung 5: Architektur des *ARToolkit* [3]

Das *ARToolkit* selbst lässt sich in drei verschiedene Teile zergliedern. Zum einen besteht es aus dem Gsub-Modul, welches für die Zeichenroutinen zuständig ist. Dieses Modul benutzt die OpenGL und GLUT Bibliotheken. In der jetzigen Version 2.71.3 kann das Gsub-Modul durch das Gsub-Lite-Modul ersetzt werden, das effizientere Algorithmen für die Grafikdarstellung beinhaltet und unabhänig von der Fensterverwaltung durch GLUT ist.
Ein weiterer Teil des *ARToolkits* ist das Video-Modul, welches für die Videoverarbeitung und Verwaltung verantwortlich ist, und auf der betriebssystemspezifischen Videolibary arbeitet.
Der letzte und der eigentliche Kernbereich ist das Augmented-Reality-Modul. Dieses beinhaltet sämtliche Routinen, die für das Tracking der Marker notwendig sind [3].

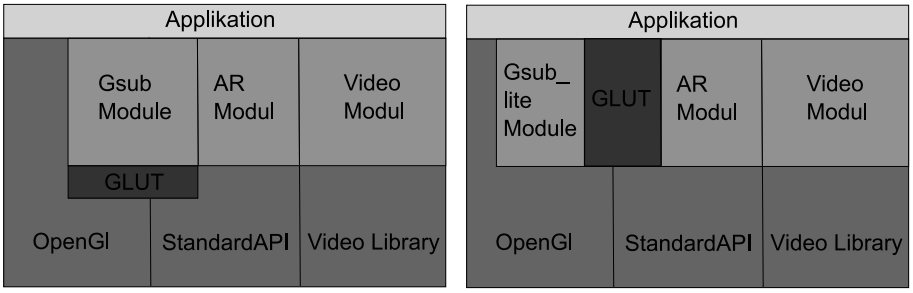

Abbildung 6: Module des *ARToolkit* [3]

Um das Zusammenspiel dieser verschiedenen Teile noch einmal zu verdeutlichen, kann man sich die drei Module in einer Art Pipeline vorstellen(Abb. 7). Das Video-Modul initialisiert die Kamera, verwaltet die relevanten Daten des Aufnahmegerätes und holt sich immer das jeweilige aktuelle Bild zur Verarbeitung. Über dieses Bild lässt das AR-Modul seine Algorithmen laufen, und liefert die Trackingdaten, wie Position und Orientierung der Marker. Mit Hilfe dieser Trackingdaten, kann dann das Gsub-Modul bzw. Gsub-lite-Modul das augmentierte Bild darstellen.

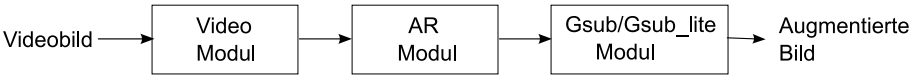

Abbildung 7: Pipelineprinzip des *ARToolkit* [3]

Im Kapitel 3.1 wird noch näher auf die Videoanbindung und Verwaltung des *ARToolkit* eingegangen. Die Funktionen des Tracking übernimmt in dem neuen Gesamtsystem das *ARToolkitPlus*, das im Abschnitt 2.3 näher betrachtet wird.

2.2.3 Eigenschaften

Wie schon im vorigen Abschnitt erwähnt, arbeitet das *ARToolkit* mit einem markerbasierten Trackingverfahren. Die zu diesem Zweck benötigten Marker sind dem System momentan noch völlig unbekannt und müssen ihm erst „antrainiert“ werden [3]. Eine kleine Auswahl an Markern zum Testen wird vom *ARToolkit* selbst bereits mitgeliefert. Sollen jedoch eine größere Menge an Markern, oder spezielle Marker benutzt werden, können diese mit Hilfe eines Blankomarkers, der nur das umgebene schwarze Quadrat beinhaltet, selbst erstellt werden. Dies kann einfach mit Hilfe eines Bildverarbeitungsprogramms geschehen. Dabei sollten keine zu komplexen oder abstrakten Symbole gewählt werden, da dadurch das Erkennen erschwert wird. Einfache, asymmetrische Zeichen eignen sich am Besten [3]. Ist der Marker erstellt, muss er dem System bekannt gemacht werden. Dies passiert mittels eines mitgelieferten Programms, welches dann eine Bitmap-Datei für den Marker erstellt. Diese Datei kann später in das Programm geladen werden, wo die detektierten Pattern mit diesem Muster verglichen werden. Dies nennt man Template-Matching Verfahren.

Ebenso wie das *ARToolkit* die Möglichkeit bietet einzelne Marker im Kamerabild zu finden, besitzt es auch eine Funktion, mit der es „Multi-Marker“ finden kann. Das bedeutet, eine Anzahl von Marker, die auf einer Fläche fest zueinander angeordnet sind. Dadurch bekommt man die Möglichkeit, eine Position zu erkennen, wenn auch nur einer dieser Marker im aufgenommenen Bild zu sehen ist [3]. Zum Beispiel befinden sich auf so einem Multimarker sechs verschiedenen Pattern. Diese werden dem System mit ihren Positionen zueinander antrainiert. Die sechs Muster stellen aber nur eine Position und Orientierung dar. Wenn jetzt nur zwei von diesen Pattern im Kamerabild zu finden sind, ist es trotzdem möglich die Position und Orientierung des Multimarkers zu bestimmen.

Als weitere Eigenschaft kann man die manuelle Angabe eines Grenzwertes beim Detektieren der Marker sehen, der die Helligkeit der Umgebung mit berücksichtigt. Dieser Grenzwert ist eine Zahl zwischen 0 und 255, und gibt an, ab welchem Mittelwert das Farbbild in ein Binärbild umgewandelt wird. Zum Beispiel muss in einem schattigem Bild der Grenzwert ziemlich niedrig gesetzt werden, um den Marker noch zu erkennen, in einem hellen Bild ziemlich hoch.

Durch die Möglichkeit der Videoanbindung unterstützt das *ARToolkit* viele verschiedene Eingabegeräte, wie die meisten gängigen USBCams, FireWire-Kameras usw. Ebenso unterstützt es die gängigsten Pixelformate, wie etwa RGB und YUV [3]. Als letztes ist noch zu erwähnen, das der gesamte Code des *ARToolkit* in C geschrieben ist, also keine klassenbasierte API zur Verfügung steht.

2.3 Das ARToolkitPlus

Nachdem das *ARToolkit* nun ausreichend vorgestellt wurde, soll an dieser Stelle auf das *ARToolkitPlus* eingegangen werden. Dieses ist sozusagen der Nachfolger des *ARToolkits*, das heißt es basiert im Grunde auf der Funktionsweise und den Algorithmen des älteren Systems, wobei aber auch verschiedene Algorithmen verändert, erweitert und ergänzt wurden. So kam zum Beispiel ein ID-Detektionsalgorithmus hinzu, der vom *ARTag* inspiriert worden ist [5]. Des Weiteren handelt es sich bei dem *ARToolkitPlus* nicht um eine Komplettlösung, wie es beim *ARToolkit* der Fall ist. Weder eine Videoanbindung, noch Algorithmen zur grafischen Darstellung werden mitgeliefert. Beim diesem AR-System handelt es sich um eine reine Bibliothek für das Tracking. Aus diesem Grund richtet sich das System auch nicht an absolute Programmieranfänger, deren Ziel nur das Ausprobieren einer einfachen kleinen Augmented Reality Anwendung ist, sondern schon an Personen, für die Programmieren nicht ein reines Fremdwort darstellt [5]. Ungeachtet dessen stellt das *ARToolkitPlus* sich hier als interessante Alternative dar.

2.3.1 Funktionsweise

Wie schon oben erwähnt, basiert die Funktionsweise im Grunde auf dem selben Prinzip wie die des *ARToolkit*, siehe Kapitel 2.2.1. Aus diesem Grund wird sie hier nicht noch einmal vorgestellt.

2.3.2 Struktur

Das *ARToolkitPlus* stellt, wie oben schon beschrieben, nur die Bibliothek mit den Funktionen und Algorithmen, die für das Tracking der Marker zuständig sind, zur Verfügung. Es fehlen die Module für die Videoanbindung und Grafikdarstellung. Da in dieser Arbeit das Tracking des *ARToolkitPlus* genutzt werden soll(vgl. Kap. 2.6), wird die Struktur dieses ARModuls an dieser Stelle näher untersucht.
Im Prinzip besteht die Bibliothek aus fünf verschiedenen Klassen. Es existieren zwar noch weitere Hilfsklassen, auf die man als Entwickler einer AR-Anwendung aber nicht zugreifen muss.

- Klasse Logger: Diese Klasse ermöglicht es, Nachrichten an die aufrufende Instanz auszugeben, wie zum Beispiel Fehlermeldungen. Es handelt sich dabei um eine abstrakte Klasse. Wenn diese in der Anwendung verwendet werden soll, so muss erst eine Implementation erfolgen.[2]

- Klasse MemoryManager: Diese Klasse kümmert sich um eine benutzerspezifische Speicherverwaltung.[2]

Die Klassen Logger und MemoryManager wurden in dieser Studienarbeit nicht näher analysiert und verwendet.

- Klasse Tracker: Diese Klasse ist die Hauptklasse des *ARToolkitPlus*. In dieser Klasse finden sich alle Funktionen, die auch im Original AR-Modul des ARToolkit implementiert sind.[2]
- Klasse TrackerMultiMarker,TrackerSingleMarker: Diese beiden Klassen sind von der Klasse Tracker abgeleitet. Sie beinhalten die speziellen Funktionen für das MultiMarker Tracking und das einfach Markertracking.[2]

Normalerweise wird mit Objekten der Klassen TrackerMultiMarker und TrackerSingleMarker gearbeitet.[5] Besonders die Klasse TrackerSingleMarker ist für diese Arbeit interessant. Auf das Vorstellen der einzelnen Methoden der Klasse wird hier jedoch verzichtet, da diese in der Dokumentation des *ARToolkitPlus* nachgelesen werden können. An dieser Stelle sollen nur einige grundlegende Eigenschaften und Zusammenhänge hervorgehoben werden, die aus der Dokumentation nicht ohne weiteres ersichtlich sind.

Der Aufbau der Headerdateien des *ARToolkitPlus* ist im ersten Moment komplex und verwirrend. Da gibt es nicht nur einfach eine *Tracker.h*, und jeweils eine *TrackerMultiMarker.h* und eine *TrackerSingleMarker.h*, sondern zusätzlich auch noch eine *TrackerImpl.h*, eine *TrackerMultiMarkerImpl.h* und eine *TrackerSingleMarkerImpl.h*. Den Zusammenhang dieser Dateien und Klassen untereinander soll Abbildung 8 verdeutlichen.

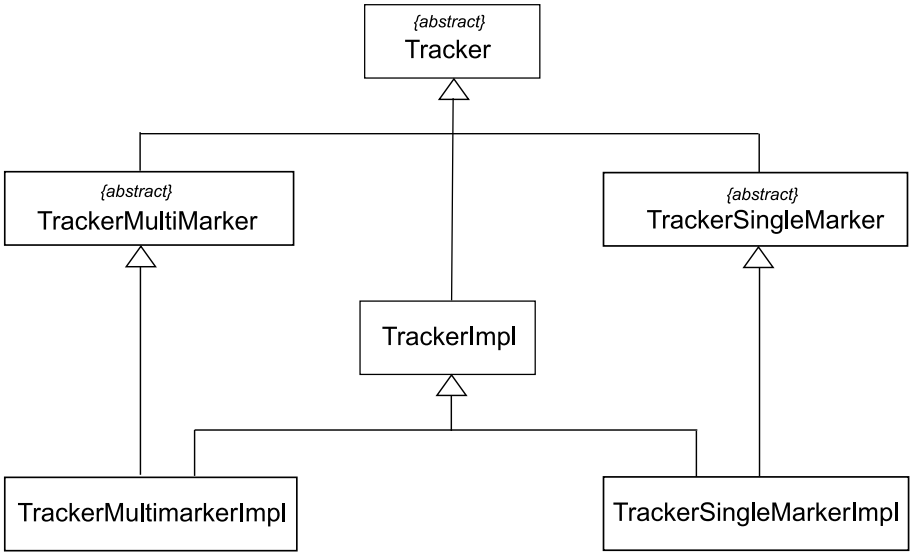

Abbildung 8: Zusammenhang der Tracker Klassen

Hierbei fungieren die einfachen Headerdateien ohne das „Impl“ als abstrakte Klassen, dass heißt von ihnen kann nur abgeleitet werden, es können aber keine Objekte von ihnen erstellt werden. Man kennt diese Klassen auch als

sogenannte Schnittstellen oder Interfaces. Sie vereinbaren für alle abgeleiteten Klassen Methoden und Funktionen, die implementiert werden müssen. Die Klassen TrackerMultiMarker und TrackerSingleMarker sind zum Beispiel von der abstrakten Klasse Tracker abgeleitet, implementieren deren Methoden jedoch nicht, und werden dadurch wieder zu weiteren abstrakten Klassen. Wenn jetzt später zum Beispiel eine weitere Klasse von TrackerSingleMarker abgeleitet wird, so muss diese Klasse die Methoden der Trackerklasse und die der TrackerSingleMarkerKlasse implementieren, es sei denn, es soll eine weitere abstrakte Klasse entstehen. Im Falle des *ARToolkitPlus* ist die Klasse TrackerSingleMarkerImpl von der Basisklasse TrackerSingleMarker und der Klasse TrackerImpl abgeleitet. Sie ist keine abstrakte Klasse. Das bedeutet, das sie alle Methoden der Basisklassen Tracker und TrackerSingleMarker implementieren muss, und die Methoden der Klasse TrackerImpl erbt. Wichtig sind für die Entwicklung von AR-Software die Klassen TrackerImpl, TrackerSingleMarkerImpl und TrackerMultiMarkerImpl. Besonders sollte dabei mit den beiden letzteren gearbeitet werden, da diese die speziellen Methoden und Funktionen für die jeweilige spezifische Anwendung besitzen. Aus diesen Klassen werden dann die benötigten Instanzen erstellt, mit denen dann weitergearbeitet werden kann.

2.3.3 Eigenschaften

Das *ARToolkitPlus* ist genau wie das *ARToolkit* markerbasiert. Das Erkennen der Marker kann beim *ARToolkitPlus* aber zusätzlich über codierte ID-Marker verlaufen, und nicht mehr nur über das Abgleichen von Mustern. Das Erkennen der Marker funktioniert, indem ein im Bild erkannter binärer Marker dekodiert wird, und man dadurch die ID erhält. Dabei kennt das System schon eine gewisse Anzahl an IDs von Markern. Hier gibt es zwei verschiedene Arten kodierter Marker, die BCH Marker und SimpleID Marker. Von den BCH Markern gibt es 4096 und von den SimpleID Markern gibt es 512 Marker die schon zur Verfügung stehen.[12] Details über die Kodierungsart findet sich auf der Webseite des *ARToolkitPlus* [5].

Auch beim *ARToolkitPlus* lässt sich die Beachtung der Helligkeitsbedingungen mit einbringen. Der Grenzwert für die Umwandlung des Bildes in ein Binärbild lässt sich, genau wie beim *ARToolkit*, manuell einstellen. Es gibt jedoch auch die Möglichkeit einer automatischen Grenzwertbestimmung. Dabei wird für die Berechnung des automatischen Grenzwertes die Situation im vorangegangenen Bild mit hinzugenommen. Sobald kein Marker mehr gefunden wird, wird per Zufall der Grenzwert solange verändert, bis ein neuer Marker detektiert wurde. Durch diesen Algorithmus ist es möglich, auch bei variierenden Lichtverhältnissen während des laufenden Programmes, die Marker zu erkennen.[12]

Des Weiteren stellt das *ARToolkitPlus* einen Algorithmus zur Verfügung, der die Eckenverschattung bei Kameras berücksichtigt. Normalerweise nimmt bei runden Kameralinsen die Helligkeit von der Mitte zu den Rändern ab. Dies ist in den binären Bildern als Verschattung zu erkennen, und wird durch den Algorithmus kompensiert, siehe Abbildung 9. Auch stellt das System einen robusteren Trackingalgorithmus zur Verfügung, als das *ARToolkit*. Dieser kann seine Effektivität jedoch noch nicht auf den mobilen Einheiten, wie PDAs oder Handys, ausschöpfen. Aus diesem Grund ist es sinnvoll, ihn nur bei der Benutzung des Systems auf Dekstoprechnern oder Notebooks zu verwenden.[12]

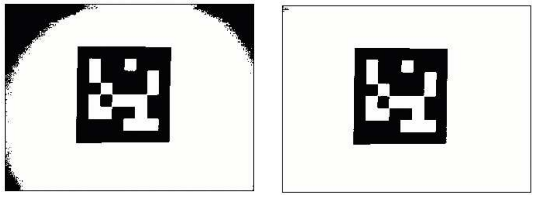

Abbildung 9: Verminderung der Eckenverschattung [12]

Dadurch dass das *ARToolkitPlus* nur die Tracking Bibliothek mitliefert, gibt es auch keine Unterstützung für verschiedene Eingabegeräte. Dafür ist es aber besonders für mobile Systeme ausgelegt, wie Handys und PDAs. Für diese Geräte ist ein Speichermanagement-System implementiert worden, auf welches in dieser Ausarbeitung nicht näher eingegangen wird. Ebenso wie das *ARToolkit* unterstützt das *ARToolkitPlus* die meisten gängigen Pixelformate. Auch sind MultiMarker, wie sie im *ARToolkit* vorgestellt wurden, möglich.

Schliesslich ist noch anzumerken, dass das *ARToolkitPlus* eine klassenbasierte API zur Verfügung stellt. Ihr Code ist in C++ geschrieben und baut auf Templates auf.

2.4 Sonstige AR-Systeme

Es gibt außer den oben vorgestellten Systemen noch eine ganze Reihe anderer AR-Systeme, die mit markerbasiertem Tracking arbeiten. Einige bekannte sind der *ARBrowser*, welcher im ARVIKA Projekt benutzt wird[13], *Cybercode*[11] und das *ARTag*[10]. Diese Systeme sollen hier aber nicht näher vorgestellt werden.

2.5 Koordinatensysteme

Für alle diese Trackingbibliotheken spielen mehrere unterschiedliche Koordinatensysteme eine Rolle. Hierzu gehören das Kamerakoordinatensystem,

das Markerkoordinatensystem, das Bildschirmkoordinatensystem des idealen Bildes und das Bildschirmkoordinatensystem des verzerrten Bildes. (Diese Koordinatensysteme sind weitgehend aus der Bildverarbeitung bekannt.) Den Zusammenhang erklärt Abb. 10 näher.

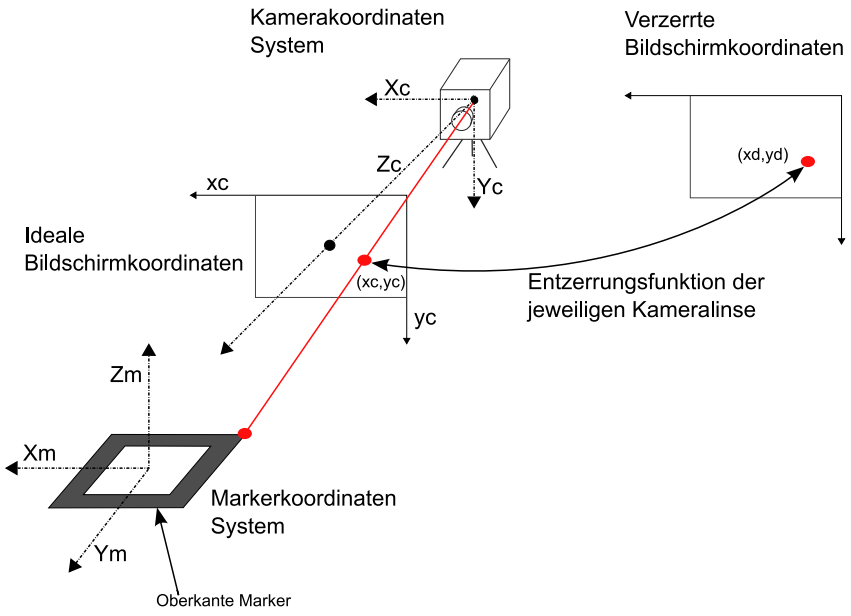

Abbildung 10: Koordinatensysteme

Im Folgenden werden die Beziehungen zwischen diesen Koordinatensystemen kurz näher erläutert. Zum einen gibt es die Abhängigkeiten zwischen der Kamera und dem Marker. Um von einem ins andere Koordinatensystem zu kommen, ist eine Rotation und eine Translation nötig. Diese Transformationen lassen sich mit Hilfe einer Matrix ausdrücken, die auch später zur Bestimmung der Position und Orientierung der virtuellen Objekte benötigt wird. Sie gibt die Position und Orientierung des Markers im Kamerakoordinatensystem an.[8] Dieser Zusammenhang lässt sich wie in Formel 1 gezeigt, ausdrücken.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix} \tag{1}$$

Des Weiteren muss man vom Kamerakoordinatensystem in das ideale Bildschirmkoordinatensystem gelangen. Für diese Umrechnung spielen Projektion und intrinsische Kameraparameter eine Rolle, also Beschaffenheiten der Kamera, die sich nicht durch die Bewegung der Kamera ändern. Diese Parameter lassen sich ebenfalls durch eine Matrix, siehe Formel (2), darstellen.[8]

Dabei stellen die Werte sf_X und sf_Y die skalierten Brennweiten dar. x_C und y_C den Versatz des Mittelpunkts des Bildschirms in Pixel.

$$\begin{bmatrix} hX_I \\ hY_I \\ h \end{bmatrix} = \begin{bmatrix} sf_X & 0 & x_C & 0 \\ 0 & sf_Y & y_C & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \tag{2}$$

Als letztes bleibt noch die Beziehung zwischen idealen und wirklich aufgenommen Bildschirmkoordinaten offen. Hierbei spielt die Verzerrung durch die Kameralinse eine Rolle, die sich für jede Kamera spezifisch bestimmen lässt. Glücklicherweise ist die Ermittlung dieser Daten nur einmal für jedes Aufnahmegerät notwendig, so dass nach einmaliger Kalibrierung die Werte für die weiteren Berechnungen zur Verfügung stehen. So kommt man mittels dieser Ergebnisse vom idealen zum real aufgenommen Bild.

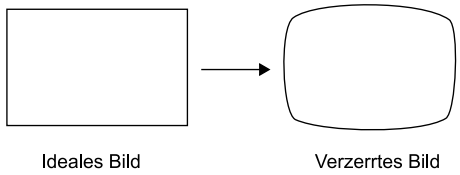

Abbildung 11: Verzerrung durch die Kameralinse

Die Bestimmung der Matrix für die Position und Orientierung der Marker beim *ARToolkit* und beim *ARToolkitPlus* wird im Folgenden kurz beschrieben. Dabei ist wichtig zu nennen, das man hier die Angaben aus nur einem Bild berechnet. Das aufgenommene, durch die Kameralinse verzerrte Bild ist der Ausgangspunkt. Dieses wird mit Hilfe der Entzerrungsfunktion der jeweiligen Kamera auf die idealen Bildschirmkoordinaten zurückgerechnet. Von dort aus werden mit Hilfe der Größe der Pattern und eines internen Algorithmus die Position und Orientierung des Markers ermittelt. Als Ergebnis erhält man die Matrix mit der Rotation und Translation, die angibt,wo sich der Marker im Kamerakoordinatensystem befindet.

2.6 Komponentenauswahl

Nach eingehender Betrachtung der hier vorgestellten Systeme, ist die Entscheidung für das Tracking auf das *ARToolkitPlus* gefallen. Da dieses System aber weder eine Videoanbindung, noch Grafikdarstellung zur Verfügung stellt, wurde entschieden, die Videoanbindung des *ARToolkit* zu separieren und sie mit dem *ARToolkitPlus* zu verbinden. Aus diesem Grund stellt sich das neue System als eine Mischung aus diesen beiden AR-Systemen dar, welche in Abschnitt 2.2 und 2.3 schon näher vorgestellt wurden. Zur Begründung: Das *ARToolkitPlus* ist im Grunde eine Weiterentwicklung des

ARToolkit und bringt somit einige Vorteile mit sich, wobei die Spezialisierung auf mobile Endgeräte in diesem Fall kein Entscheidungsfaktor gewesen ist, da als Ausführungsobjekt ein Notebook vorgesehen war (vgl.Abschnitt1.3). Für das *ARToolkitPlus* haben hingegen insbesondere die automatische Grenzwertbestimmung, das robustere Tracking, welches nur auf dem PC arbeitet[12] und die Id-basierte Markererkennung gesprochen. Als Anforderung stand im Raum, dass das System robust gegen Helligkeitsunterschiede sein müsse, da in der Filmszene die Beleuchtung eine wichtige Rolle spielt, so gibt es beispielsweise extrem beleuchtete oder auch verschattete Bereiche. Hier wäre eine manuelle Einstellung nicht möglich, da dabei das Programm jedes Mal neu gestartet oder die Einstellung manuell betätigt werden müsste, was während einer Kamerafahrt nicht sehr sinnvoll ist. Da hier das *ARToolkitPlus* die Möglichkeit einer automatischen Anpassung des Grenzwertes bietet, ist es in diesem Punkt klar vorzuziehen. In Abb. 12 und Abb. 13 ist der Vorteil eindeutig erkennbar. In beiden Fällen wurde während des laufenden Programmes der Marker verschattet. Beim *ARToolkit* führte dies dazu, dass der Marker nicht mehr erkannt wurde.

Abbildung 12: ARToolkit links ohne Verschattung rechts mit Verschattung

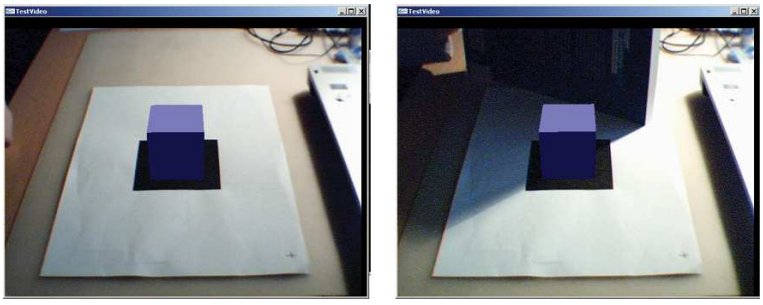

Abbildung 13: ARToolkitPlus links ohne Verschattung rechts mit Verschattung

Des Weiteren sollte es möglich sein, mehrere Marker in dem System zu nutzen, da in einer Filmszene zum Beispiel auch mehrere Objekte eine Rolle spielen. Der große Vorteil beim *ARToolkitPlus* liegt hierbei darin, dass dem

System nicht erst mühsam Marker „antrainiert“ werden müssen, sondern dass das System schon ein großes Repertoire an Markern besitzt, die es schon kennt. Weiterhin wird das Template-Matching Verfahren des *ARToolkit* bei mehreren Markern immer langsamer, weil jeder erkannte Marker mit jedem geladenen Muster verglichen werden muss. Beim *ARToolkitPlus* erfolgt das ohne Geschwindigkeitssverlust, da die erkannten Marker mit Hilfe eines Dekodierungsverfahrens erkannt und die erhaltene ID einfach auszugeben ist.
Ein weiterer Vorteil liegt darin, dass das *ARToolkitPlus* klassenbasiert implementiert wurde, so das es kein Problem darstellte dieses auch objektorientiert zu erweitern. Das schafft mehr Übersichtlichkeit durch die Kapselung.

Die Mehrzahl der anderen Systeme, die oben kurz angesprochen wurden, standen von vornherein nicht zur Auswahl, da sie für kommerzielle Zwecke gedacht sind und sich daher nicht mit diesem Projekt vereinbaren ließen. Aus diesem Grund kamen letztlich nur drei Systeme in die nähere Auswahl, das *ARToolkit*, das *ARToolkitPlus* und das *ARTag*. Diese AR-Systeme stehen für nicht-kommerzielle Nutzung zur freien Verfügung(GNU Lizenz)[4]. Auch das *ARTag* ist mit in die Überlegung für das neue System eingeflossen, vor allem weil es den Vorteil bietet, dass auch beim teilweise Verdecken des Markers, dieser trotzdem noch eindeutig erkannt werden kann. Trotzdem ist die Entscheidung auf das *ARToolkitPlus* gefallen, da hier einfach eine bessere Dokumentation durch das *ARToolkit* vorhanden war, und schon einige Erfahrungen mit dem *ARToolkit* an der Universität gemacht worden sind. Ebenso machte das *ARTag* beim Ausprobieren des Testprogramms einen sehr instabilen Eindruck. Aus diesen Gründen wurde es nicht mehr in die weitere Planung mit einbezogen.

3 Implementation

Das neue Gesamtsystem entstand in der Enwicklungsumgebung „Microsoft Visual Studio.Net 2003“ und wurde unter dem Betriebsystem Windows XP auf einem Notebook entwickelt.

3.1 Videoanbindung

Wie schon im Kapitel 2.2.2 erwähnt, wird vom *ARToolkit* nur das Modul der Videoanbindung benötigt. Da dieses in der Programmiersprache C geschrieben ist, das *ARToolkitPlus* aber die Programmiersprache C++ benutzt, wurde eine Wrapper Klasse geschrieben, die einen einheitlichen Zugriff möglich macht, und den Vorteil der Kapselung mit sich bringt. In dieser Studienarbeit wird dabei nur die Videoanbindung unter dem Betriebssystem Windows XP betrachtet.

Um das Video-Modul aus dem *ARToolkit* zu isolieren, müssen zuerst einmal Abhängigkeiten der Module des Systems untereinander untersucht und aufgehoben werden. Dafür wird das Framework des *ARToolkit* näher betrachtet. In diesem befinden sich sieben relevante Projekte, aus denen nachher die für die Beispiele und Applikationen notwendigen Bibliotheken erstellt werden. Nachfolgend werden diese kurz vorgestellt.

- libAR: beinhaltet die für das Tracking notwendigen Algorithmen und ist damit die Kernlibrary des Frameworks
- libARgsub, libARgsub-lite, libARgsubUtil: beinhalten die Routinen für die Grafikdarstellung
- libARMulti: ist für die Verarbeitung von Multimarkern zuständig. Mit Multimarker ist ein Markerfeld gemeint, wobei mehrere Marker eine Position und Orientierung darstellen
- libARvrml: hiermit wird die Virtual Reality Modeling Language[4] mit eingebunden.
- libARvideo: ist für die Videoanbindung zuständig, aus diesem Grund für diese Arbeit die entscheidende.

3.1.1 Die libARvideo

Die Bibliothek libARvideo beinhaltet Funktionen, die man für das Initialisieren, das Öffnen und Schließen der Videoquelle und das Holen eines Bildes

[4] VRML ist eine Beschreibungssprache für 3D Szenen.

benötigt. Um das Funktionieren der Videoanbindung unter dem Betriebssystem Windows zu gewährleisten, wird die DSVideolib (Direct Show Video Processing Library)[1] von Thomas Pintaric benutzt. Ebenfalls muss DirectX 9.0 oder höher installiert sein. In Abb. 14 sieht man die Zusammenhänge zwischen diesen Systemen. Die libARvideo des *ARToolkit* benutzt die DsVideolib, die sich um die Kommunikation mit den Kameratreibern kümmert. Sie arbeitet mit xml-Dateien, worüber manuell bestimmte Konfigurationen der Kamera angegeben werden können. Diese Videobibliothek arbeitet mit der DirectX-Runtime zusammen, welche als Schnittstelle zur Kamera fungiert.

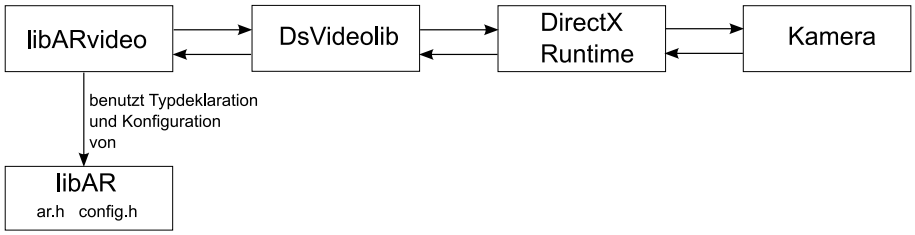

Abbildung 14: Verbindungen der libARvideo

3.1.2 Abhängigkeiten der libARvideo von anderen ARToolkit Libraries

Die drei Module, die das *ARToolkit* vereint, sind weitgehend unabhängig voneinander, bis auf ein paar Kleinigkeiten. Es benutzt zum Beispiel seine eigenen Typdefinitionen. Diese sind in der ar.h deklariert, die Teil der libAR ist, werden aber auch von dem Video-Modul benötigt. Ebenfalls benutzt die libARvideo die config.h der libAR. Diese Definitionen und Konfigurationen müssen im Gesamtsystem durch die Definitionen und Konfigurationen des *ARToolkitPlus* ersetzt werden.

3.1.3 Neue Klassen

Wie schon oben erwähnt, soll hier eine Art Wrapperklasse erzeugt werden, die die passenden Funktionen kapseln und so den ganzen Vorgang übersichtlicher gestalten soll. Um das Modul der Videoanbindung aus dem Toolkit zu isolieren, wurde als erstes eine neue Projektmappe erstellt, in der sich nur noch die libARvideo befand. Wichtig war hierbei, dass alle Pfade der include- und library-Dateien angepasst und richtig verlinkt wurden. Die Abhängigkeiten von der libAR wurden vorläufig mit in die libARvideo eingebunden. Danach wurde ein neues Projekt erstellt, mit den neuen Klassen „CamVideo“und „Image“, die im Folgenden näher beschrieben werden.

Klasse Image
Die Klasse Image ist für die Verwaltung des Videobildes zuständig, das von der Klasse CamVideo von der Kamera abgefragt wird. Dabei ist sie so konzipiert das Höhe, Breite und Pixelformat des Bildes mit abgespeichert werden, und so alle wichtigen Informationen des Bildes in dem Objekt gespeichert sind. Für die Abfrage des Pixelformats musste noch eine weitere Funktion in der libARvideo implementiert werden.

Klasse CamVideo
Die Klasse CamVideo stellt Funktionen für das Öffnen und Schließen der Videoquelle zur Verfügung. Ebenfalls erstellt diese Klasse ein Objekt vom Typ Image, in dem das derzeitige Videobild mit Höhe, Breite und Pixelformat abgespeichert wird. Dieses kann über die Funktion „getCamImage“abgerufen werden.

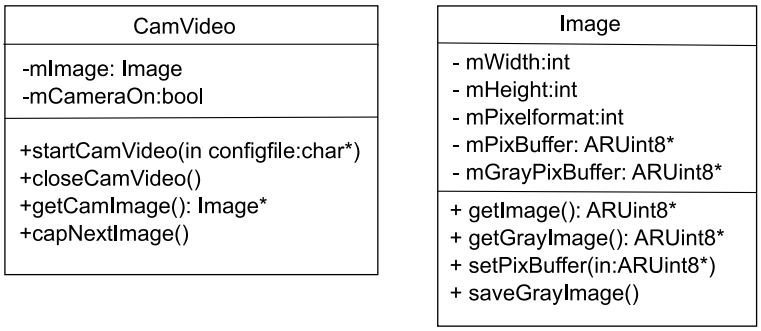

Abbildung 15: Neue Klassen der Videoanbindung

3.2 Das Tracking mit dem ARToolkitPlus

Nachdem die Entscheidung, das *ARToolkitPlus* für das Tracking im neuen System zu nutzen, gefallen war, musste dieses noch passend eingebunden werden. Dafür ist ein Kompilierungsvorgang des Toolkits notwendig, so wie das Einbinden der entsprechenden Headerdateien der Tracker Klasse, die man benötigt. Wichtig ist es hierbei auch, wieder alle Pfade zu der Bilbiothek und den Include-Dateien richtig zu setzen. Nachdem alle Abhängigkeiten beachtet wurden, kann das *ARToolkitPlus* im neuen System genutzt werden.

3.2.1 Neue Klassen

Das Ziel, welches mit dieser Arbeit erreicht werden sollte, war es, beliebig viele Marker in Abhängigkeit zu einem Weltmarker zu detektieren, und sich ausgeben zu lassen. Da es diese Funktion in der *ARToolkitPlus* Bibliothek so nicht gibt, wurde hierfür eine neue Klasse mit dem Namen TrackerMultiSingleMarkerImpl erstellt. Diese ist von der Klasse TrackerSingleMarkerImpl

abgeleitet, und wurde zu den üblichen Methoden um zwei erweitert. Weiterhin wurde eine Klasse „TrackCoord“angelegt. Für jeden Marker, der im Kamerabild gefunden wurde, wird ein Objekt vom Typ dieser Klasse erstellt, und so zur Informationsabfrage bereit gestellt.

Klasse TrackCoord
Die Klasse TrackCoord kümmert sich prinzipiell um die Verwaltung der einzelnen Marker.Für jeden wird ein Objekt dieser Klasse angelegt, welches die jeweilige ID, Größe und Zentrum des Markers enthält. Ebenfalls werden die Matrix in welcher die Position und Orientierung des Markers im Kamerakoordinatensystem und die Matrix in welcher die Position und Orientierung des Markers im Weltmarkerkoordinatensystem mit abgespeichert.

Klasse TrackerMultiSingleMarkerImpl
Die Klasse TrackerMultiSingleMarkerImpl stellt zu den bereits in der TrackerSingleMarkerImpl implementierten Methoden zwei Weitere zur Verfügung. Zum ersten eine Methode, die alle initialen Einstellungen vornimmt, mit denen der Anwender nicht belastet werden soll. Des Weiteren eine Methode, die den normalen Trackingsalgorithmus darin erweitert, das auch mehrere einzelne Marker detektiert und diese zueinander in Beziehung gesetzt werden können. Die letztere Methode gibt einen STL-Vector zurück, in dem jeder Marker, der im Bild erkannt wurde, als TrackCoord Objekt abgespeichert wird. Mit Hilfe dessen können die relevanten Daten leicht ausgelesen werden.
Wie schon im vorigen Kapitel erwähnt, wird beim *ARToolkitPlus* ein ID-basiertes Detektierverfahren benutzt. Aus diesem Grund kennt das System schon eine bestimmte Anzahl an Markern. In dieser Arbeit wurden die „Simple ID-Marker“ mit der Standardrandbreite 0,25 der Gesamtbreite benutzt. Dabei nimmt der Marker mit der ID = 0 eine Sonderstellung ein, da er als Weltmarker festgelegt wurde, zu dem alle anderen Marker in Relation gesetzt werden.

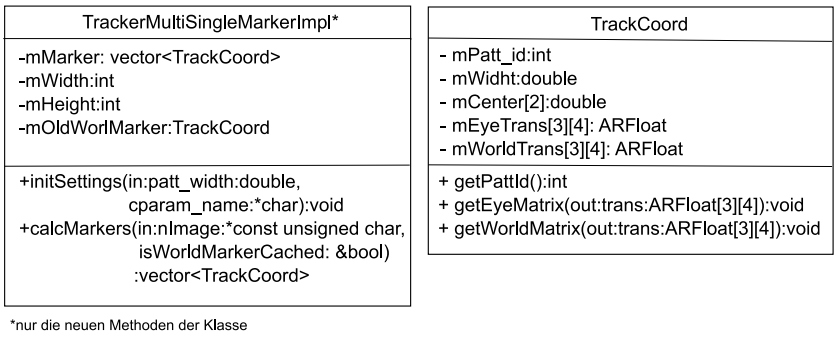

Abbildung 16: Neue Klassen des Trackings

3.2.2 Die Methode *calcMarkers(const unsigned char* nImage)*

Diese Methode befindet sich in der Klasse TrackerMultiSingleMarkerImpl und enthält den Hauptalgorithmus, der für das Detektieren von mehreren Markern zueinander und zu einem Weltmarker verwendet wird. Das Listing 1 verdeutlicht die Vorgehensweise.

Listing 1: Methode für das Detektieren mehrerer Marker zueinander

```
1  std::vector<TrackCoord>& calcMarkers(const unsigned char* nImage){
2  ARMarkerInfo            *marker_info;
3  int                     marker_num,worldMarker = -1;
4  TrackCoord              a;
5  ARFloat                 iTrans[3][4], wTrans[3][4], camTrans[3][4],
6                          wMarkTrans[3][4];
7
8  if(nImage==NULL)
9  return mMarker;
10
11 PROFILE_BEGINSEC(profiler, SINGLEMARKER_OVERALL)
12 confidence = 0.0f;
13
14 if(arDetectMarker(const_cast<unsigned char*>(nImage), this->thresh,
15                         &marker_info, &marker_num) < 0)
16 {
17         PROFILE_ENDSEC(profiler, SINGLEMARKER_OVERALL)
18         return mMarker;
19 }
20 for (int i=0; i < marker_num; i++)
21 {
22         if(marker_info[i].id == 0)
23         {
24                 mMarker.clear();
25                 executeSingleMarkerPoseEstimator(&marker_info[i],
26                         patt_center,patt_width, wTrans);
27                 arUtilMatInv(wTrans,camTrans);
28                 a = TrackCoord(marker_info[i].id, patt_width,
29                         wTrans, camTrans);
30                 mMarker.push_back(a);
31                 worldMarker = 0;
32         }
33 }
34 if (worldMarker > -1)
35 {
36         for(int j=0; j < marker_num; j++)
37         {
38                 if(marker_info[j].id == 0)
39                 {
40                         continue;
41                 }
42                 if(marker_info[j].id == -1)
43                 {
44                         continue;
```

```
45          }
46                  executeSingleMarkerPoseEstimator(&marker_info[j],
47                          patt_center,patt_width, iTrans);
48                  arUtilMatMul(camTrans,iTrans,wMarkTrans);
49                  a = TrackCoord(marker_info[j].id, patt_width,
50                          iTrans, wMarkTrans);
51                  mMarker.push_back(a);
52          }
53  }
54  else
55  {
56          mMarker.clear();
57  }
58  return mMarker;
59  }
```

In Zeile 14 und 15 wird das übergebene Bild nach Markern durchforstet. Werden welche gefunden, so werden diese mit Ihrer ID in der Variable „*marker_info“ gespeichert. In „marker_num“ befindet sich die Anzahl der detektierten Marker. Sollte das Detektieren fehlschlagen, wird ein leerer STL-Vector zurückgeliefert. Die nächsten Zeilen behandeln den Weltmarker. Hier wird der Marker mit der ID = 0, aus dem Repertoire des *ARToolkitPlus* von „SimpleId“, als Weltmarker festgelegt. Dabei wird die maximale Anzahl an detektierten Markern durchlaufen und nachgesehen, ob sich der Weltmarker unter den gefundenen Markern befindet. Sollte dieser Durchlauf positiv sein, so wird in Zeile 25 die Position und Orientierung des Markers berechnet und als Matrix in „wTrans“ abgespeichert. Mit Zeile 27 wird die Inverse von „wTrans“ in „camTrans“ gespeichert. Mit der Matrix „camTrans“ hat man nun die Position und Orientierung der Kamera im Weltmarkerkoordinatensystem. Siehe Abbildung 17.

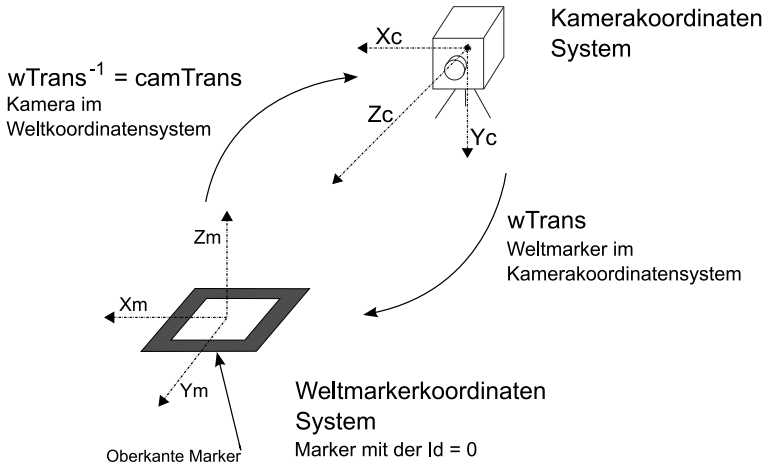

Abbildung 17: Beziehung zwischen Weltmarker und Kamera

Nach diesen Schritten wird ein Objekt vom Typ TrackCoord erstellt, in dem die beiden Matrizen und die ID abgespeichert werden. Der Weltmarker kommt als erstes in den STL-Vector, der an das aufrufende Programm zurückgeliefert wird. Sollte kein Weltmarker gefunden werden, wird ein leerer STL-Vector an das Programm zurückgeliefert. Sobald ein Weltmarker gefunden wurde, werden die restlichen gefundenen Marker durchlaufen. Siehe Zeile 36 bis 52. Für jeden Marker wird die Position des Markers im Kamerakoordinatensystem berechnet, und dann mit Hilfe der Matrix des Weltmarkers die Position jedes Markers zum Weltmarkerkoordinatensystem. Siehe Abbildung 18.

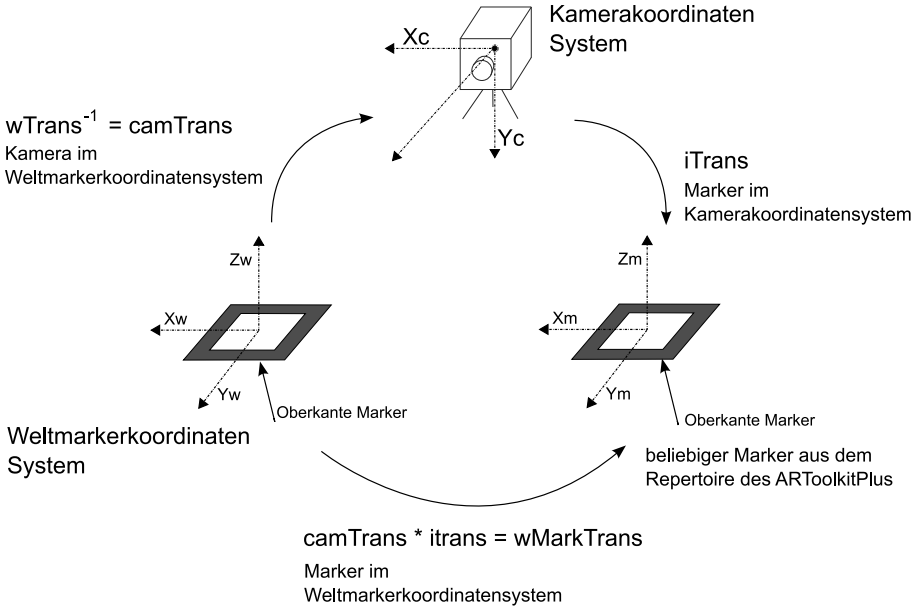

Abbildung 18: Beziehung zwischen Markern und Kamera

Für jeden Marker wird ein Objekt vom Typ TrackCoord erstellt und in dem STL-Vector abgespeichert.
Eine weitere wichtige Anforderung war es, wenn der Weltmarker nicht im Bild zu sehen wäre, die alte Position und Orientierung des Weltmarkers benutzt werden sollte. Hierfür wurde die Methode von dem Team (Claus Weymann, Andreas Gutsche), welches auf diesem System eine Maya Application aufbaute, wie folgt erweitert (Die Änderung wurde durch dieses Team vorgenommen, da dieses zeitgleich mit der Entwicklung ihres Programmes begonnen hatte, das diese Eigenschaft schon benötigte. Die Veränderungen wurden weitgehend in dieser Arbeit mit übernommen.)

Listing 2: Erweiterte Methode für das Detektieren mehrerer Marker zueinander

```
vector<TrackCoord>& calcMarkers(const unsigned char* nImage,
bool &isWorldMarkerCached){
ARMarkerInfo    *marker_info;
int             marker_num;
TrackCoord      a;
ARFloat         iTrans[3][4], wTrans[3][4], camTrans[3][4],
                wMarkTrans[3][4];


if(nImage==NULL)
return mMarker;

PROFILE_BEGINSEC(profiler, SINGLEMARKER_OVERALL)
confidence = 0.0f;

if(arDetectMarker(const_cast<unsigned char*>(nImage), this->thresh,
                  &marker_info, &marker_num) < 0)
{
        PROFILE_ENDSEC(profiler, SINGLEMARKER_OVERALL)
        return mMarker;
}

mMarker.clear();
for (int i=0; i < marker_num; i++)
{
        if(marker_info[i].id == 0)
        {
                executeSingleMarkerPoseEstimator(&marker_info[i],
                        patt_center,patt_width, wTrans);
                arUtilMatInv(wTrans,camTrans);
                a = TrackCoord(marker_info[i].id, patt_width,
                        wTrans,camTrans);
                ARFloat eyeMatrix[3][4];
                a.getEyeMatrix(eyeMatrix);
                if(eyeMatrix[0][0] == 0)
                {
                        //if found world marker is [[0 0 0 0]...] then
                        //pretend that no world marker was found
                        continue;
                }
                mMarker.push_back(a);
                mOldWorldMarker = a;
                isWorldMarkerCached = false;
        }
if (mMarker.empty())    //Änderung
{
        mMarker.push_back(mOldWorldMarker);
        isWorldMarkerCached = true;
}

for(int j=0; j < marker_num; j++)
```

```
52 {
53         if(marker_info[j].id == 0)
54         {
55                 continue;
56         }
57         if(marker_info[j].id == -1)
58         {
59                 continue;
60         }
61         executeSingleMarkerPoseEstimator(&marker_info[j],
62                 patt_center,patt_width, iTrans);
63         arUtilMatMul(camTrans,iTrans,wMarkTrans);
64         a = TrackCoord(marker_info[j].id, patt_width,
65                 iTrans,wMarkTrans);
66         mMarker.push_back(a);
67 }
68
69 return mMarker;
70 }
```

Zu Anfang wird eine neue Variable „isWorldMarkerCached“ im Funktionsrumpf eingeführt, die dem aufrufenden Programm angibt, ob ein Weltmarker erkannt wurde oder nicht. Des Weiteren wird eine neue Variable „mOldWorldMarker“ angelegt, welche die Daten des vorherigen Weltmarkers enthält, und dann wieder aufgerufen werden kann, wenn kein Weltmarker gefunden wurde. Der Durchgang funktioniert ähnlich wie im Programmsegment oben. Erst wird nach dem Weltmarker gesucht, sollte dieser gefunden werden, beginnt derselbe Ablauf wie in der ersten Version. Wird keine Weltmarker gefunden, wird in Zeile 46 der alte Weltmarker als aktueller Weltmarker angenommen und die Cache-Variable auf „true“ gesetzt. Danach werden die weiteren erkannten Marker durchlaufen, und in Relation zu dem alten Weltmarkerkoordinatensystem berechnet.

3.2.3 Die Methode *initSettings(double width char *cparam)*

In der Methode „initSettings“ werden die initialen Konfigurationen vorgenommen, die für das Arbeiten mit dem *ARToolkitPlus* nötig sind. Sie befindet sich ebenfalls in der Klasse TrackerMultiSingleMarkerImpl.

Listing 3: Methode initSettings

```
1 void initSettings(double width, char *cparam_name)
2 {
3 this->setPixelFormat(ARToolKitPlus::PIXEL_FORMAT_BGRA);
4
5 if(!this->init(cparam_name, 1.0f, 1000.0f))
6 {
7         printf("ERROR:␣init()␣failed\n");
8         delete this;
```

```
9           system("Pause");
10          exit(0);
11  }
12  this->changeCameraSize(mWidth,mHeight);
13
14  this->setPatternWidth(patt_width);
15
16  this->setBorderWidth(0.250f);
17  this->setMarkerMode(ARToolKitPlus::MARKER_ID_SIMPLE);
18
19  this->activateAutoThreshold(true);
20  this->setUndistortionMode(ARToolKitPlus::UNDIST_LUT);
21  this->setPoseEstimator(ARToolKitPlus::POSE_ESTIMATOR_RPP);
22  }
```

Zu Anfang wird hier das Pixelformat gesetzt. PIXEL_FORMAT_BGRA bedeutet, dass ein Bild mit Farbinformationen übergeben wird, dass heißt, es werden 32 Bits pro Pixel benötigt, jeweils acht für einen Kanal. Danach wird die Kamerakalibrierungsdatei „cparam_name“ geladen, in der sich die spezifischen Kameraparameter, wie zum Beispiel die Verzerrung der Kameralinse befinden. In Zeile 12 wird die Bildgröße der Kamera neu gesetzt. Dies ist notwendig, da in der *ARToolkitPlus* Bibliothek diese Größe zu Anfang schon auf 320x240 fest gesetzt ist. Wenn jetzt der Anwender zum Beispiel zu Beginn eine andere wählt, und „CameraSize“ nicht neu gesetzt würde, werden keine Marker im Bild erkannt, da die internen Berechnungen falsch ablaufen.
Wichtig für die internen Berechnungen ist auch die richtige Angabe der Größe der Pattern, da mit Hilfe dieser die Position und Orientierung der Marker im Kamerakoordinatensystem berechnet wird. Abbildung 19 soll verdeutlichen, was passiert, wenn bei einer angebenen Patterngröße von 80mm ein Marker mit der Größe 40mm detektiert wird.

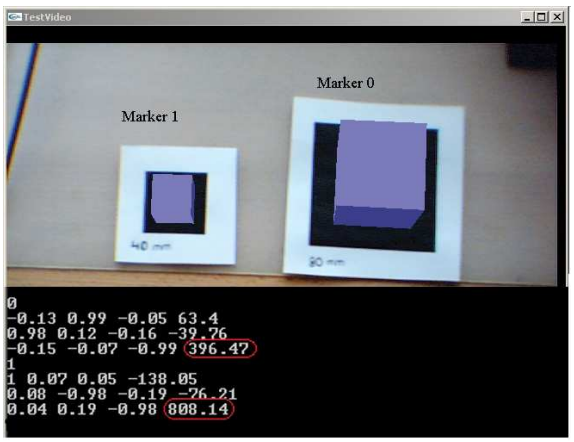

Abbildung 19: Richtige Größe der Marker

Es ist eindeutig zu erkennen, dass der linke Marker, also der Kleinere, als weiter entfernt angenommen wird. Dies ist auch aus dem Entfernungswert „z“(rotumrandet) zur Kamera festzustellen. Der linke Marker soll doppelt so weit entfernt sein, wie der Rechte. Da die beiden annähernd in der selben Ebene liegen, ist das eine Fehlinformation.
Die nächsten beiden Zeilen Code geben an, also Zeile 16 und 17, welcher Typ von Marker benutzt wird. In diesem Fall handelt es sich um den Typ Marker Simple-Id mit der Randbreite 0,25.
In Zeile 19-21 werden die Möglichkeiten, die das *ARToolkitPlus* gegenüber dem *ARToolkit* mehr bietet, eingestellt. Dabei handelt es sich um die automatische Grenzwerterkennung und den robusteren Berechnungsalgorithmus für die Position und Orientierung. Diese wurden schon im Kapitel 2.3.3 vorgestellt.
Der Anwender muss im Hauptprogramm nur die Größe der Pattern und die Kamerakalibrierungsdatei angeben. Alle anderen Einstellungen sollen nicht von ihm verändert werden.

4 Ergebnisse

Ziel dieser Arbeit war es, ein Trackingsystem für ein Previsualisierunsgstool, welches in Maya und Touch realisiert werden soll, zu erstellen(vgl. Abschnitt 1.3). Der Endzweck dieses Werkzeuges soll es sein, die Planung von Filmszenen für alle Akteure zu erleichtern.
Das Ergebnis stellt nun ein Gesamtsystem dar, welches die Vorteile der Videoanbindung des *ARToolkits* bereit stellt und diese mit den Vorzügen des *ARToolkitPlus* verbindet. Die Wahl dieser Systeme wurden im Kapitel 2.6 schon näher begründet. Ebenfalls wurde die Funktionalität des *ARToolkitPlus* so erweitert, dass das Tracking von mehreren Markern möglich wird, und diese in Bezug zu einem Weltkoordinatensystem, welches wiederum durch einen Marker repräsentiert wird, stehen.

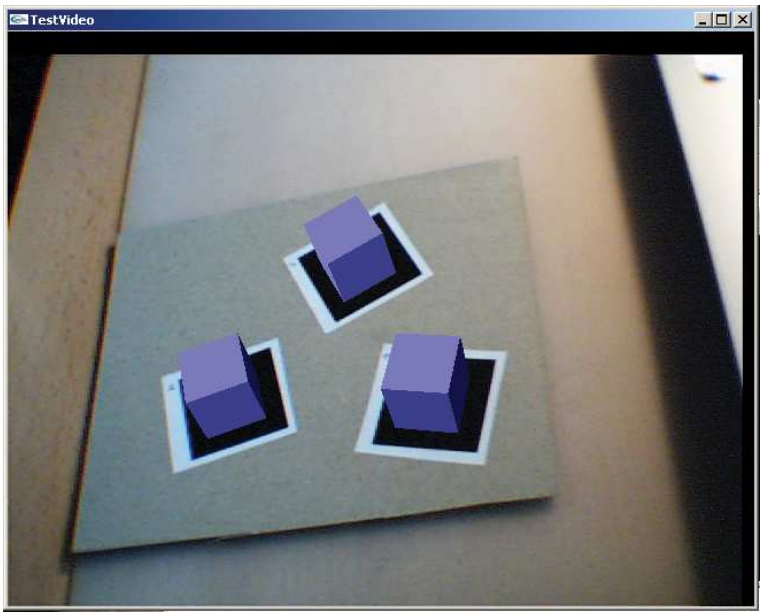

Abbildung 20: Tracking mehrerer Marker

Für die Realisierung des Systems wurde der Code des *ARToolkits*[9], entwickelt von Dr. Hirokazu Kato von der Osaka University in Japan, und der Code des *ARToolkitPlus*[5] von Daniel Wagner von der Technischen Universität Graz, benutzt. Die Nutzung dieser Systeme ist unter der GNU General Public License (GPL) [4] freigegeben, sie stehen also für nicht kommerzielle Zwecke zur freien Verfügung. Für die Kommunikation der Treiber mit dem Toolkit ist eine aktuelle Version von *DirectX* notwendig. Ebenso wird für die Videoanbindung des *ARToolkit* eine Version der *DirectShow Video Processing Library* [1] (Version 0.0.8) von Thomas Pintaric von der Vienna University of Technology benötigt.

Abbildung 21: Tracking mehrerer Marker auf verschiedenen Ebenen

Die in dieser Ausarbeitung vorgestellte Lösung wurde auf einem Notebook entwickelt und ist für das Betriebssystem Windows XP ausgelegt. Des Weiteren können die meisten herkömmlichen USB-Webcams und Firewirekameras benutzt werden. Getestet wurden eine USB-Webcam der Firma Trust und die Firewire Webcam Fire-i der Firma Unibrain, wobei es bei der Fire-i Probleme mit den Treibern gab, so dass das Schließen des Anwendungsfensters nur über die Taste „Esc“ funktioniert. Dafür liefert diese aber durch den Anschluss über Firewire eine höhere Bildfrequenz.
Damit das System funktioniert fehlt noch eine xml-Datei für die Anbindung der Kamera, in der die dafür benötigten Informationen enthalten sind. Diese befindet sich mit im Projektordner, und kann gegebenenfalls erweitert werden. Ebenfalls benötigt wird eine Kalibrierungsdatei der jeweiligen Kamera. Diese sollte grundsätzlich mit dem Kalibrierungstool des *ARToolkits* erstellt werden können. Leider musste festgestellt werden, dass das *ARToolkitPlus* diese Datei in dieser Form nicht annahm. Aus diesem Grund wurde vorläufig mit der Standardkalibrierungsdatei des *ARToolkit* gearbeitet. Glücklicherweise liefert diese annähernd gute Bilder, und kann so für weitere Testprogramme genutzt werden.

4.1 Installation

Bei dem hier realisierten Gesamtsystem handelt es sich, wie beim *ARToolkit*, um eine Sammlung von Bibliotheken, auf welche die AR-Anwendung

verlinkt. Alle hierfür notwendigen Dateien befinden sich im Ordner „ProjektAR“.
Um die Bibliotheken nutzen zu können, müssen diese im ersten Schritt kompiliert werden. Es wird empfohlen für die Kompilierung „Microsoft Visual Studio .NET 2003“ zu verwenden, da dieses Programm auch in der Entwicklung zum Einsatz gekommen ist und somit mögliche Veränderungen von Einstellungen umgangen werden können.
Für das Erstellen der Bibliotheken wird die „ProjektAR.sln“ im obersten Ordnerverzeichnis geöffnet und die Projektmappe erstellt. Die kompilierten Bibliotheken befinden sich im Ordner „ProjektAR/lib“, die des *ARToolkitPlus* in „ProjektAR/ARToolkitPlus_2_1_1/lib/win32“. Dies ist wichtig zu wissen, wenn eigene Anwendungen auf die Bibliotheken verlinken sollen.
Zum Test des Systems wurde ein Beispielprogramm implementiert, welches sich im Ordner „ProjektAR/samples“ befindet. Da dieses ebenfalls mit in die Projektmappe eingebunden ist, liegt nach dem Kompilierungsvorgang der Bibliotheken ein ausführbare Exe-Anwendung im Ordner „ProjektAR/bin“ vor. Zum Ausführen des Programms werden noch Marker benötigt. Diese finden sich in dem Ordner „ProjektAR/id-markers/simple/std-border“. Diese Marker können in einer beliebigen Größe ausgedruckt werden, wobei die Voreinstellung im Programm 80mm beträgt. Wird eine andere Größe gewählt, ist es wichtig das diese angpasst wird, siehe Abschnitt 3.2.3. Der Marker mit der ID = 0 ist der Weltmarker, und somit notwendig, um das Weltkoordinatensystem zu setzen. Die anderen Marker können beliebig gewählt werden. Als letzter Schritt muss nur noch eine Kamera angeschlossen und das Programm „ProjektAR/bin/CamVideo.exe“ gestartet werden.

4.2 Benutzung der Bibliothek CameraARlib

Die Benutzung der neuen Bibliothek soll an dieser Stelle an einem einfachen Beispielprogramm vorgestellt werden.

4.2.1 Die Videoanbindung

Listing 4: Nutzung der Videoanbindung

```
1  #include "CamVideo.h"
2
3  char                    *vconf = "Data\\WDM_camera_flipV.xml";
4
5  CamVid::CamVideo*       camera;
6  CamVid::Image*          image;
7
8  void init( void )
9  {
10         camera = new CamVid::CamVideo();
11
```

```
12          camera->startCamVideo(vconf);
13  }
14
15  void mainLoop(void)
16  {
17          image = camera->getCamImage();
18
19          camera->capNextImage();
20  }
```

Listing 4 verdeutlicht die Nutzung der Videoanbindung. Zu Beginn wird die Headerdatei „CamVideo.h“ eingebunden. Damit stehen die Funktionalitäten der Klassen *CamVideo* und *Image* zur Verfügung. Nach dem Anlegen jeweils eines Objekts vom Typ *CamVideo* und *Image* wird in Zeile 12 die Videositzung gestartet, das bedeutet die Verbindung zur Videoquelle ist gelegt. Die zu diesem Zweck verwendete xml-Datei enthält alle nötigen Daten, die hierfür benötigt werden. Mit der Methode „getCamImage()“ werden im Objekt „image“ die Bilddaten des aktuellen Bildes wie Höhe, Breite, Pixelformat und das Bild selber abgespeichert. Dieses kann damit für die weitere Verwendung im Tracking benutzt werden. Sobald sichergestellt ist, dass das aktuelle Bild nicht mehr benötigt wird, kann mit Hilfe von „capNextImage()“ ein neues Bild angefordert werden.

4.2.2 Das Tracking

Im Listing 5 wird ein Beispiel für die Benutzung des Trackings der neuen Bibliothek gegeben.

Listing 5: Nutzung des Trackings

```
1   #include "TrackerMultiSingleMarkerImpl.h"
2
3   CamVid::TrackerMultiSingleMarkerImpl *tracker;
4
5   void init( void )
6   {
7   tracker = new CamVid::TrackerMultiSingleMarkerImpl
8   (camera->getWidth(),camera->getHeight());
9
10          tracker->initSettings(80.0, cparam_name);
11  }
12
13  void mainLoop(void)
14  {
15
16  int                     id;
17  ARFloat                 eyetrans[3][4], worldtrans[3][4];
18
19  bool isWorldMarkerCached;
```

```
20
21 std::vector<CamVid::TrackCoord>  &markers = tracker->calcMarkers
22                 (image->getImage(),isWorldMarkerCached);
23
24 if (!markers.empty())
25         {
26         for (int i=0; i< markers.size() ; i++)
27                 {
28                         markers[i].getEyeMatrix(eyetrans);
29                         markers[i].getWorldMatrix(worldtrans);
30                         id = markers[i].getPattId();
31                 }
32 }
```

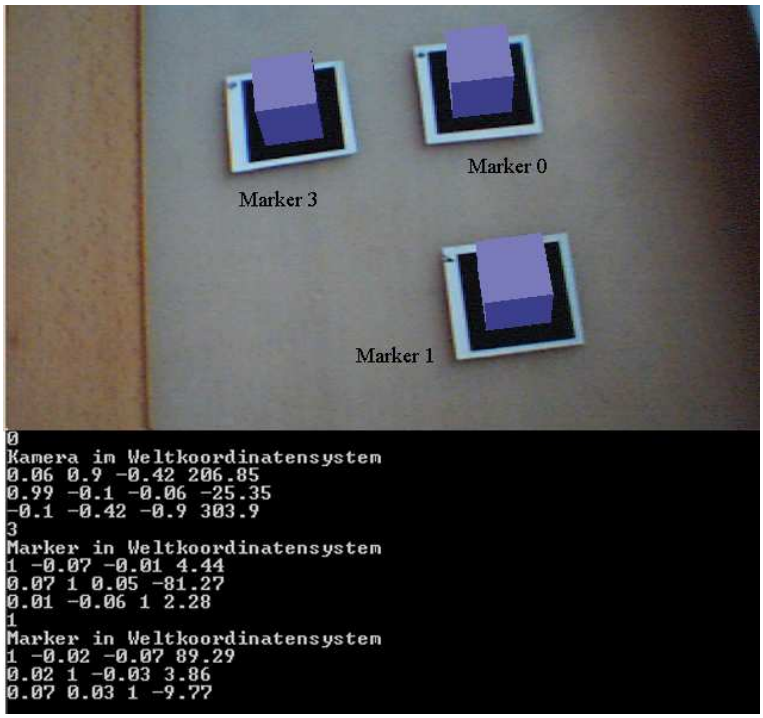

Abbildung 22: Tracking mehrerer Marker mit Ausgabe der Weltmatrizen

Auch für das Tracking muss eine Headerdatei eingefügt werden, und zwar die „TrackerMultiSingleMarkerImpl.h“. Der Name steht für das Tracken mehrerer einzelner Marker. Nachdem auch hier wieder ein Objekt dieser Klasse angelegt wurde, wird es zuerst mit der richtigen Bildgröße erstellt. Darauf hin werden einige Grundeinstellungen vorgenommen. Der Benutzer muss nur die Größe seiner ausgedruckten Marker angeben und die für seine Kamera spezifische Kalibrierungsdatei. In dem Projekt ist eine Standarddatei eingebunden, mit der das Tracken bereits mit verschiedenen Kameras gut funktioniert. Die anderen Einstellungen wurden schon im Abschnitt 3.2.3 näher

erläutert, wobei hier prinzipiell auch keine Veränderungen vorgenommen werden müssen. In der Hauptschleife befindet sich das eigentlich Tracking. Mit der Methode „calcMarkers“ wird das übergebene Bild nach Markern durchsucht. Die gefundenen Marker werden als *TrackCoord* Objekte im *STL-Vector* „markers“ gespeichert und können damit abgerufen werden. Das Abfragen der Informationen kann man durch die in Zeile 28-30 befindlichen Befehle erreichen. Mit „getEyeMatrix(eyetrans)“ wird die berechnete Transformationsmatrix im Format 3x4 in „eyetrans“ gespeichert, die den Marker im Kamerakoordinatensystem angibt. Mit „getWorldMatrix(worldmatrix) “ wird die Matrix in „worldmatrix“ gespeichert, die den Marker im Weltmarkerkoordinatensystem angibt und mit „getPattId()“ kann die ID abgefragt werden. Einzige Ausnahme stellt der Marker mit der ID=0 dar, er ist als erstes im *STL-Vector* gespeichert und gibt mit „worldmatrix“ die Kamera im Weltmarkerkoordinatensystem an.

5 Fazit und Ausblick

5.1 Fazit

Die Ziele dieser Studienarbeit wurden im Wesentlichen erreicht. Es wurde eine statische Bibliothek erstellt, in der die geforderten Funktionalitäten, wie die Videoanbindung an das *ARToolkitPlus* und das Tracken von mehreren Markern in einem Weltkoordinatensystem bereit gestellt wurden. Ebenfalls sind die Anforderungen, dass das neue System auf einem Notebook unter dem Betriebssystem Windows laufen sollte, erfüllt worden.

Nach anfänglichen Schwierigkeiten beim Installieren des *ARToolkits*, was auf Probleme mit den Videotreibern, dem DirektX-Treiber und einem installierten Codecpaket auf dem benutzten System zurückzuführen war, machte die Entwicklung des Gesamtsystems große Fortschritte. So war das Extrahieren der Videoanbindung aus dem *ARToolkit* mittels einer Wrapperklasse schnell geschehen. Die nächsten Probleme tauchten beim Verlinken der Videoanbindung mit dem *ARToolkitPlus* auf. Doch auch diese Verzögerungen konnten durch Recherche in Online Foren und intensiverer Beschäftigung mit der Entwicklungsumgebung Microsoft Visual Studio. NET 2003 gelöst werden. Beim Implementieren der zusätzlichen Funktionen für das Tracking stellte das *ARToolkitPlus* eine interessante Herausforderung, da diese hauptsächlich mit Templates realisiert wurden, sodass die abgeleitetet Klasse „TrackerMultiSingleMarkerImpl“ komplett in der Headerdatei implementiert werden musste. Für die aufgetreten Linkerfehler, wenn es eine Headerdatei und eine Cppdatei gab, konnte leider noch keine Lösung gefunden werden.
Zusammenfassend wurden große Fortschritte und ein großer Lerneffekt erzielt. Durch die intensive Beschäftigung mit den AR-Systemen *ARToolkit* und *ARToolkitPlus* konnte ein System entwickelt werden, welches die Verwendung des *ARToolkitPlus* unter Windows stark vereinfacht und einen leichten Einstieg ermöglicht.

5.2 Ausblick

Für die Erweiterung des System gibt es natürlich immer noch viele Möglichkeiten. Zum Beispiel wäre es sicherlich nützlich, das System dahingehend zu erweitern, dass es auch auf einem Linux oder Mac Rechner funktioniert. Des Weiteren könnte eine Funktion implementiert werden, die es erlaubt, ein MultiMarkerTracking auf verschiedenen Ebenen zu realisieren. Das bedeutet, dass eine Position, am sinnvollsten die Position des Weltmarkers, über mehrere Marker, die einen festen Abstand zueinander haben, erkannt werden kann. So wäre die Position und Orientierung des Weltkoordinatensystems nicht an einem einzigen Marker gebunden, sondern würde schon erkannt werden, wenn nur einer dieser Marker sichtbar wäre. Der Unterschied zu dem bereits im *ARToolkitPlus* vorhandenen MultiMarkerTracking

läge darin, dass die Marker nicht in einer Ebene zueinander liegen würden, sondern sie sich auch zum Beispiel in der Höhenlage unterscheiden könnten. Interessanter aber werden eher die Programme sein, die auf dieses System aufsetzen. Da diese Studienarbeit innerhalb eines Projektes mit der Ludwigsburger Filmakademie entstanden ist, existieren bereits die ersten Ansätze dazu. Zum Einen wird eine Anwendung mit dem Programm Touch in Ludwigsburg entwickelt, zum anderen eine an der Universität Koblenz mit dem Programm Maya. Diese beiden Anwendungen haben das Ziel, ein Previsualisierungwerkzeug für die Filmindustrie zu entwickeln. Mit Hilfe der Marker soll es möglich sein, Objekte, die diesen Markern zugeordnet werden können, in den jeweiligen Programmen zu bewegen. Somit können Szenen schon im Voraus am Computer durchgespielt werden, damit Kameramänner, Beleuchtungstechniker usw. sich auf die jeweilige Filmsequenz besser einstellen können.

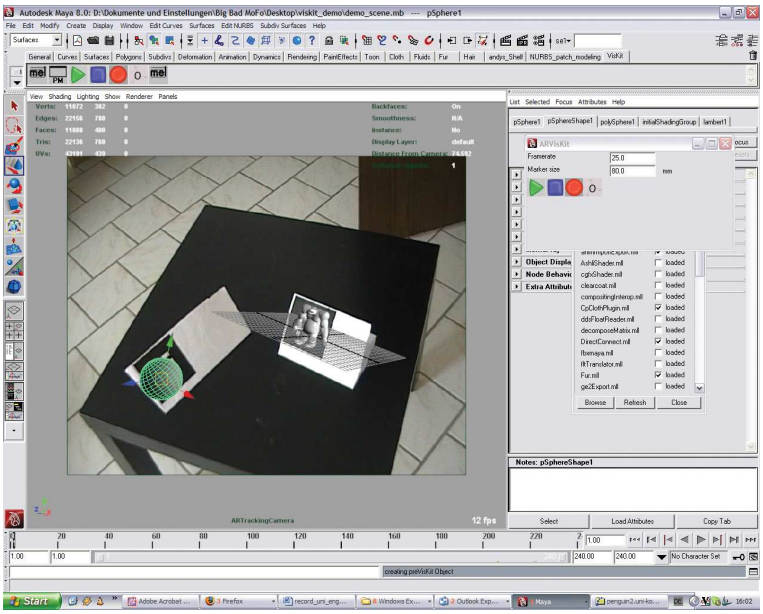

Abbildung 23: Previsualisierung mit Maya an der Universität Koblenz

Literatur

[1] http://www.sourceforge.net/projects/dsvideolib.

[2] ARToolKitPlus Documentation. http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus_docu/index.html. [Online, Oktober 2006].

[3] Dokumentation des ARToolkit. http://www.hitl.washington.edu/artoolkit/documentation/. [Online, Oktober 2006].

[4] GNU Free Documentation Licenses. http://www.gnu.org/licenses/fdl.html. [Online, März 2007].

[5] Wagner Daniel. Handheld augmented reality-ARToolkitPlus. http://studierstube.icg.tu-graz.ac.at/handheld---ar/artoolkitplus.php. [Online, Dezember 2006].

[6] Müller Stefan Prof. Dr. Virtuelle Realität und Augmented Reality. http://www.uni-koblenz.de/FB4/Institutes/ICV/AGMueller/Teaching/WS0506/VRAR. Vorlesungsunterlagen.

[7] Frauenhofer IGD. Matris. http://www.ist-matris.org. [Online, März 2007].

[8] Kato Hirokazu. Inside ARToolKit. http://www.hitl.washington.edu/artoolkit/Papers/ART02-Tutorial.pdf [Online, Dezember 2006].

[9] Philip Lamp. Webseite des ARToolkit. http://www.hitl.washington.edu/artoolkit/. [Online, Oktober 2006].

[10] Fiala Mark. ARTag. http://www.artag.net. [Online, Dezember 2006, aktualisiert Februar 2007].

[11] Rekimoto Jun, Ayatsuka Yuji. Cybercode: Designing Augmented Reality Environments with visual Tags. in Proceeding of Designing Augmented Reality Environments 2000. http://www.vi.is.u-tokyo.ac.jp/~takeo/courses/2005/media/papers/cybercode-dare2000.pdf[Online, März 2007].

[12] Wagner Daniel, Schmalstieg Dieter. ARToolkitPlus for Pose Tracking on Mobile Devices. in Proceeding of 12th Computer Vision Winter Workshop. http://www.icg.tu-graz.ac.at/Members/daniel/Publications/ARToolkitPlus/CVWW07---final.pdf; note =.

[13] Friedrich Wolfgang(Hrsg.). *ARVIKA Augmented Reality für Entwicklung, Produktion und Service*. Publicis Corporate Publishing, Erlangen, 2004.