UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Fraunhofer Institut Angewandte Informationstechnik

Echtzeitfähige Schatten in Mixed Reality-Umgebungen

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Jens Freiling

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dr. Wolfgang Broll Fraunhofer Institut Angewandte Informationstechnik

Koblenz, im September 2006

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig 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)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit
Herr Jens Freiling
(Matrikel-Nr. 201110047)

Thema: **Echtzeitfähige Darstellung realitätsnaher Schatten in VR- und AR- Umgebungen**

.

Im Rahmen dieser Arbeit soll ein echtzeitfähiger Ansatz für Schatten in einer Mixed-Reality-Umgebung entwickelt und umgesetzt werden. Dazu sollen sowohl die gängisten Schattenalgorithmen untersucht werden, als auch verschiedene Verbesserungen, wie Anti-Aliasing, Soft Shadows oder die Verwendung der Graphikhardware. Soweit die Echtzeitfähigkeit die Verbesserungen zulässt, sollen sie in den Ansatz miteinbezogen werden.

Ziel dieser Arbeit ist es, nach einer ausgiebigen Analyse der bestehenden Verfahren, eine geeignete Methode zu entwickeln und diese als Teil eines Mixed-Reality Frameworks zu implementieren. Entscheidend dabei ist die Echtzeitfähigkeit in Szenen, die sowohl für AR als auch für VR repräsentativ sind.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Untersuchung bekannter Schattenverfahren für sowohl AR als auch VR Umgebungen
2. Einarbeitung in das MORGAN Framework
3. Entwicklung einer geeigneten Testszene
4. Implementierung eines geeigneten Schattenverfahrens für sowohl AR als auch VR Umgebungen unter Ausnutzung von GPU-Shadern
5. Dokumentation der Ergebnisse

Die Diplomarbeit wird am Fraunhofer-Institut für Angewandte Informationstechnik in Sankt Augustin durchgeführt.

Betreuer: Dr. Wolfgang Broll (Fraunhofer FIT, St. Augustin)

S. Müller

Koblenz, den 1. April 2006 - Prof. Dr. Stefan Müller-

Kurzfassung

Schatten erhöhen sichtbar den Realitätsgrad von gerenderten Bildern. Außerdem unterstützen sie den Benutzer in der Augmented Reality beim Greifen und Manipulieren von virtuellen Objekten, da sie das Einschätzen von Position und Größe dieser Objekte leichter machen. 1978 veröffentlichte Lance Williams den Shadow Mapping-Algorithmus, der einen Schatten in virtuellen Umgebungen erstellt. Diese Diplomarbeit stellt einen Modifikation des Standard Shadow Mapping-Algorithmus vor, der zusätzlich in Augmented/Mixed Reality-Umgebungen genutzt werden kann. Der Ansatz erweitert den Standard Algorithmus zunächst um einem PCF-Filter. Dieser Filter behandelt das Aliasing-Problem und erstellt außerdem weiche Schattenkanten. Damit der Schattenalgorithmus aber einen Schatten in einer Mixed Reality-Umgebung erstellen kann, werden Phantomobjekte benötigt. Diese liefern dem Algorithmus die Position und die Geometrie der realen Objekte. Zur Erstellung der Schatten geht der Ansatz folgendermaßen vor: Zuerst zeichnet der Algorithmus das Kamerabild. Danach wird eine Shadow Map mit allen virtuellen Objekten erstellt. Beim Rendern der virtuellen Objekte wird mit dem Shadow Mapping ein Schatten von allen virtuellen Objekten auf sich selbst und auf allen anderen virtuellen Objekten erzeugt. Danach werden alle Phantomobjekte gerendert. Der Fragmentshader führt wieder den Tiefentest durch. Liegt ein Fragment im Schatten, so bekommt es die Farbe des Schattens, ansonsten wird die Transparenz auf eins gesetzt. Damit werden alle Schatten von den virtuellen auf den realen Objekten erzeugt. Die Ergebnisse des Ansatzes zeigen, dass dieser in Echtzeit in Mixed Reality-Umgebungen genutzt werden kann. Außerdem zeigt ein Vergleich mit einem modifizierten Shadow Volume-Algorithmus, der ebenfalls für Mixed Reality-Umgebungen genutzt werden kann, dass der eigene Ansatz einen realistischer wirkenden Schatten in kürzerer Zeit erzeugt. Somit erhöht der Ansatz den Realitätsgrad in Augmented Reality-Anwendungen und hilft dem Benutzer bei der besseren Einschätzung von Distanzen und Größen der virtuellen Objekte.

Abstract

Shadows add a level of realism to a rendered image. Furthermore, they support the user of an augmented reality application through the interactions of virtual objects. The reason for this is that shadows make it easier to judge the position and the size of a virtual object. In 1978, Lance Williams published the shadow mapping algorithm with the aim to render a shadow of objects in a virtual scene. This master thesis presents a modified shadow mapping approach that can additionally be used in Augmented/Mixed Reality applications. First of all the standard algorithm ist extended by a PCF-filter. This filter is used to handle the aliasing-problem on the edges of the shadow and also to soften the shadow. Phantom objects are necessary to be able to operate this approach in a Mixed Reality application. These objects simulate the position and the geometry of the real objects for the algorithm. The approach consists of three steps: First the camera image is drawn into the framebuffer. After that a shadow map, of the virtual objects only, is created. When rendering these objects shadow mapping creates the shadows of virtual objects onto other virtual objects and on themselves. Afterwards the phantom objects are rendered. The depth test is performed on the fragment shader. If a fragment lies in a shadowed region it will get the color of the shadow. However, if it is beeing lit its transpareny value will be set to 1 so that it will not be seen. By applying this procedure all shadows from the virtual objects onto the real objects will be drawn. The results show that the approach can be used in real time in Mixed Reality environments. Additionally a comparison with a modified version of a shadow volume algorithm that can also be used for Mixed Reality applications shows that the approach of this master thesis casts a more realistic shadow in a shorter period of time. All in all this approach increases the level of realism in augmented reality applications and it helps the user measure distances and sizes of the virtual objects more easily.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Eines der wichtigsten Ziele der 3D-Computergrafik besteht darin, Szenen auf dem Computer mit hohem Realitätsgrad darzustellen. Dabei zählt nicht einzig und allein das Aussehen der einzelnen Objekte in einer Szene. Wichtig ist die Wirkung der gesamten Szene auf den Betrachter. Um diese Wirkung zu erzielen, dürfen Schatten nicht fehlen. In der Realität wirft jedes einzelne Objekt einen Schatten, und Szenen ohne Schatten wirken nicht reell, sondern flach und künstlich.

Eine komplett virtuelle Szene (Virtual Reality), die auf dem Monitor angezeigt wird, liefert dem Betrachter nur eine 2D-Darstellung, bei der die Tiefeninformation fast komplett verloren geht. Daher ist es wichtig, sehr viele sekundäre Tiefeninformationen in das Ausgabebild zu integrieren. Das menschliche Auge orientiert sich zur korrekten Wahrnehmung und Interpretation einer Szene an den Schatten von Objekten. Die Schatten dienen dazu, dem Betrachter die Position, die Größe und die Geometrie des schattenwerfenden Objektes und die Geometrie des Schattenempfängers besser erkennen zu lassen. Also kann der Betrachter mithilfe von Schatten viel besser einschätzen, wo genau sich ein Objekt im Raum befindet, wie groß es in Relation zu den anderen Objekten ist, oder ob es auf dem Boden steht oder schwebt.

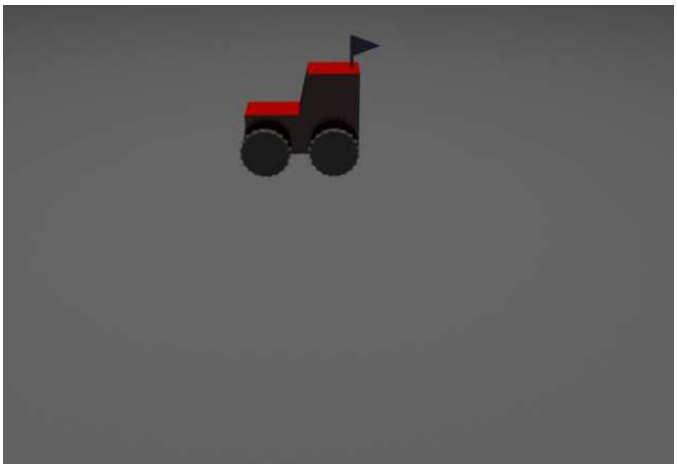


Abbildung 1: Virtuelle Szene ohne Schatten

Die Abbildungen 1 und 2 demonstrieren diese Wirkung: In Abbildung 1 ist eine virtuelle Szene mit einem Auto auf einem Untergrund zu sehen. Es ist sehr schwer einzuschätzen, wo genau sich das Auto in der Szene befindet. Es könnte sowohl weiter hinten direkt auf dem Untergrund stehen, als auch weiter vorne mit einem gewissen Abstand über dem Untergrund schweben. Also ist es schwer zugleich die Position und die Größe des Autos einzuschätzen. In Abbildung 2 ist das gleiche Bild noch einmal ganz links abgebildet. Die Bilder in der Mitte und rechts zeigen das gleiche Bild mit einem unterschiedlichen Schatten. In diesen Bildern ist für den Betrachter sofort ersichtlich, wo genau sich das Auto befindet und wie groß es ist.

Eine noch wichtigere Bedeutung haben Schatten allerdings in reellen Szenen, die mit virtuellen Objekten unterstützt werden (Augmented Reality). Der Betrachter schaut hierbei

z. B. durch ein semitransparentes Display, mit der er die reelle Welt wahrnimmt, während zugleich virtuelle Objekte eingeblendet werden. In diesem Szenario haben Schatten zwei Aufgaben: Zunächst tragen sie wiederum zum Realitätsgrad der Szene bei. Die virtuellen Objekte sollen sich so mit der realen Welt vermischen, dass der Betrachter sie nicht von den realen Objekten unterscheiden kann. Dazu sollten sie einen Schatten werfen, der sich nicht von dem eines reellen Objektes unterscheiden lässt. Ist dies nicht der Fall, erkennt der Betrachter den Unterschied zu den reellen Objekten. Dadurch ist er in seiner Wahrnehmung gestört und taucht nicht vollständig in die teilweise virtuelle Welt ein. Da der Benutzer außerdem die Möglichkeit hat, die virtuellen Objekte zu greifen und zu manipulieren, ist es die zweite Aufgabe der Schatten in einer AR-Umgebung, ihn dabei zu unterstützen. Und nur Schatten, die das genaue Einschätzen von Position und Größe der virtuellen Objekte ermöglichen, erlauben präzise Interaktionen.

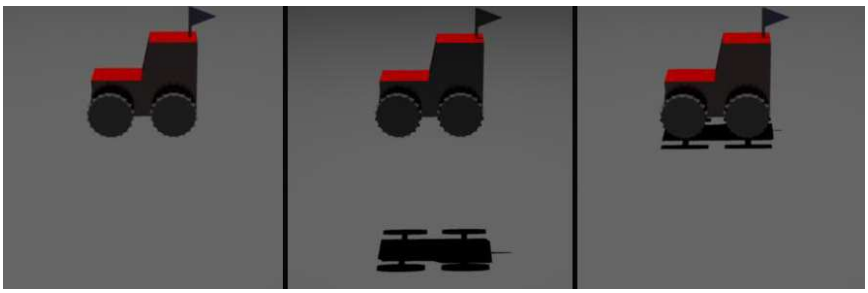


Abbildung 2: Virtuelle Szene mit Schatten: Es fällt leichter die Größe und die Position des Objektes einzuschätzen

Da das Thema echtzeifähige Schatten in Augmented Reality-(AR)-Umgebungen noch nicht ausreichend erforscht wurde, sind die oben angesprochenen Punkte bisher nicht zufriedenstellend umgesetzt worden. Daher muss auf diesem Gebiet noch viel geforscht werden, um dem Benutzer eine virtuelle Welt auch wirklich als reell erscheinen zu lassen.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll ein echtzeitfähiger Ansatz für Schatten in einer Mixed-Reality-Umgebung entwickelt und umgesetzt werden. Dazu sollen sowohl die gängigsten Schattenalgorithmen als auch verschiedene Verbesserungen untersucht werden, wie Anti-Aliasing, Soft Shadows oder die Verwendung der Grafikhardware. Soweit die Echtzeitfähigkeit die Verbesserungen zulässt, sollen sie in den Ansatz miteinbezogen werden. Ziel dieser Arbeit ist es, nach einer ausgiebigen Analyse der bestehenden Verfahren, eine geeignete Methode zu entwickeln und diese als Teil eines Augmented Reality-Frameworks zu implementieren. Entscheidend dabei ist die Echtzeitfähigkeit in Szenen, die sowohl für AR als auch für VR repräsentativ sind.

Ziel der Diplomarbeit *Echtzeitfähige Schatten in Mixed Reality-Umgebungen* ist es, einen echtzeitfähigen Ansatz für Schatten in einer Mixed Reality-Umgebung zu entwickeln und umzusetzen. Das bedeutet, dass sich eine Person mit einem Head-Mounted-Display in einem reellen Raum bewegen kann, und ihr zu allen zusätzlich virtuell eingeblendeten Objekten der dazugehörige Schatten in Echtzeit generiert wird. Diese Schatten sollten dabei so reell wie möglich aussehen und sich in relativ komplexen Szenen in Echtzeit darstellen lassen.

Das Ziel beinhaltet also zunächst, unter Berücksichtigung der vorhandenen Arbeiten, die Erstellung eines echtzeitfähigen Ansatzes von Schatten sowohl in relativ komplexen VR- als auch in AR-Szenen. Wichtig dabei ist, dass die generierten Schatten keine Aliasing-Effekte oder ähnliche unschöne Artefakte aufweisen, so dass sie für den Betrachter reell wirken. Außerdem erstrebenswert ist das Verbessern der Schatten in sogenannte Soft-Shadows, also weichen Schatten, die den Realitätsgrad der Szene erheblich erhöhen. Dieser Ansatz soll des Weiteren in einer Mixed-Reality-Umgebung umgesetzt werden.

1.3 Gliederung

Die Diplomarbeit *Echtzeitfähige Schatten in Mixed Reality-Umgebungen* wird in Kapitel 2 zunächst eine Einführung in die Grundlagen der für die Arbeit wichtigen Themengebiete geben. Im Einzelnen umfasst dies die Themen Mixed Reality, Echtzeitfähigkeit und die verschiedenen Arten von Schatten, wie sie zunächst in der Natur auftreten und dann in der Computergrafik simuliert werden können.

Kapitel 3 stellt bisherige Arbeiten zum Thema Schattenalgorithmen in Echtzeit im Überblick vor. Dabei handelt es sich hauptsächlich um die beiden wichtigsten Verfahren *Shadow Maps* und *Shadow Volumes* und um eine Vielzahl an Erweiterungen. Des Weiteren werden einige Ansätze für AR-Umgebungen vorgestellt, die zu großen Teilen auf einem der beiden Schattenverfahren aufbauen. Kapitel 4 behandelt einige der bisherigen Arbeiten im Detail. Hierbei handelt es sich um Arbeiten, die für die Konzeption des Ansatzes dieser Diplomarbeit berücksichtigt worden sind.

Kapitel 5 bis 7 beinhalten den Ansatz dieser Diplomarbeit. Dafür wird in Kapitel 5.2 analysiert, welche Anforderungen das System haben sollte und welche speziellen Probleme bzw. Fragestellungen dabei auftauchen werden. Der restliche Teil des Kapitel 5 stellt das Konzept des Ansatzes vor. Hier werden verschiedene Algorithmen und ihre Erweiterungen aufgegriffen, die in den Ansatz mit einbezogen werden könnten. Kapitel 6 erläutert die Umsetzung des vorgestellten Ansatzes und beschreibt, welche Konzepte übernommen wurden. In Kapitel 7 werden die Ergebnisse des Ansatzes präsentiert. Entscheidend sind hierbei vor allem die Geschwindigkeit und das Aussehen. Im letzten Schritt wird der eigene Ansatz mit einem modifizierten Shadow Volume Algorithmus für die Augmented Reality verglichen.

Kapitel 8 fasst die wesentlichsten Aspekte des vorgestellten Ansatzes noch einmal zusammen. Des Weiteren enthält der Ausblick einige Möglichkeiten, wie der Ansatz verbessert werden könnte und beschreibt einige Anwendungsszenarien, in denen er einsetzbar wäre.

Im Anhang dieser Diplomarbeit befinden sich außerdem einige Auszüge aus dem Quellcode des Ansatzes und die Quellenangaben.

2 Grundlagen

Schon der Titel *Echtzeitfähige Schatten in Mixed Reality-Umgebungen* enthält Begriffe, deren Bedeutung zunächst geklärt werden muss. Das Kapitel Grundlagen erläutert die wichtigsten Begriffe aus den Themen Mixed Reality, Echtzeitfähigkeit und Schatten.

2.1 Mixed Reality

Mit dem Begriff Mixed Reality (deutsch: vermischte Realität) werden Umgebungen oder Systeme zusammengefasst, in denen sich die reelle (physische) Welt mit der virtuellen Welt vermischt. Diese Vermischung erlaubt verschiedene Klassifizierungen, die im Virtual Continuum [35] zusammengestellt sind.

1. Die *reale Umgebung* ist die Realität. Sie beinhaltet keine virtuellen Elemente.
2. Eine Umgebung, in der virtuelle Objekte zu einer realen Szene hinzugefügt werden, wird *Augmented Reality* (AR) genannt.
3. *Augmented Virtuality* (AV) ist eine Umgebung, bei der die Szene virtuell ist und mit reellen Objekten ausgestattet wird.
4. Die letzte Umgebung enthält gar keine reellen Elemente mehr und wird daher *Virtual Reality* (VR) genannt.

Da das Hauptaugenmerk dieser Arbeit auf der Erstellung von Schatten in AR und VR Umgebungen liegt, werden diese beiden Begriffe in den folgenden Kapiteln etwas ausführlicher beschrieben.

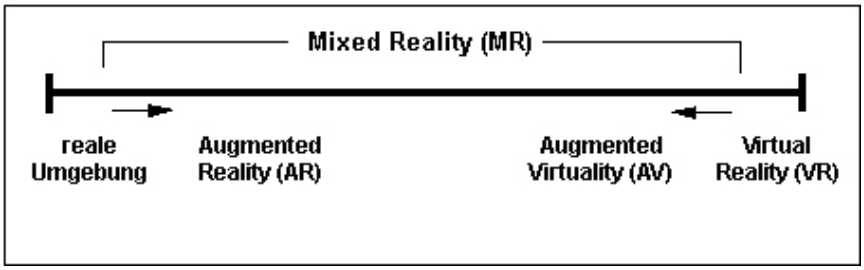


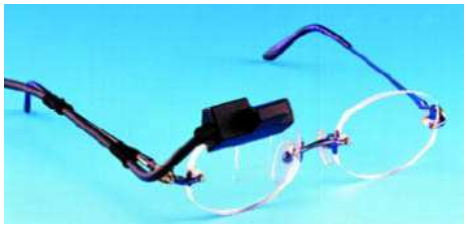
Abbildung 3: Mixed Reality Continuum [36]

2.1.1 Augmented Reality (AR)

Augmented Reality (AR) bedeutet im Grunde „erweiterte Realität“. Sie blendet dem Benutzer virtuelle Informationen ein, die mit seinem realen Sichtfeld überlagert werden. Diese virtuellen Informationen unterstützen den Benutzer bei der Interaktion mit den realen Objekten. Beispielsweise könnten einem Monteur eines technischen Gerätes Teile des Gerätes virtuell angezeigt werden, die er nicht sehen kann, weil sie z. B. hinter einer Wand liegen. Für die Darstellung von Augmented Reality gibt es zwei verschiedene Möglichkeiten:

1. Optisches See-Through
2. Video-See-Through

Beim optischen See-Through (Abbildung 4(a)) setzt der Benutzer eine Brille mit semitransparentem Display auf. Er nimmt die Realität direkt wahr und die virtuellen Informationen werden zusätzlich eingeblendet. Dieses Display funktioniert im Grunde wie eine normale Brille. Dadurch gibt es keine Probleme bezüglich der Simultanität und der Videobandbreite beim Aufnehmen eines Kamerabildes und des natürlichen Stereosehens. Allerdings können Latenzen beim Einblenden der virtuellen Informationen entstehen. Außerdem muss eine Kalibrierung vorgenommen werden, so dass ein Verrutschen der Datenbrille eine Neukalibrierung erforderlich machen kann. Der wohl größte Nachteil des optischen See-Through hängt damit zusammen, dass die virtuellen Objekte über das Sichtfeld des Benutzers gelegt werden. Legt man zum Beispiel ein virtuelles Objekt über ein reales, so sollte das reale Objekt darunter nicht mehr sehen sein können. Ist das virtuelle Objekt allerdings nicht groß genug oder zum Teil transparent, so kann es passieren, dass das reale Objekt an bestimmten Stellen immer noch zu sehen ist, obwohl es dort nicht mehr zu sehen sein dürfte. Dies sind die so genannten Geisterbilder [40, 50].



(a) optisches See-Through [46]

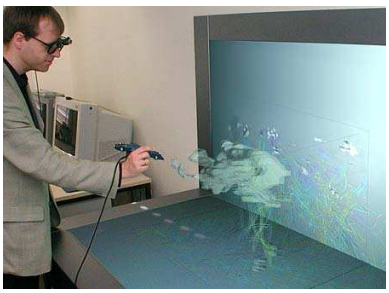
(b) Video-See-Through

Abbildung 4: Head-Mounted-Displays

Beim Video-See-Through (Abbildung 4(b)) wird die Realität mithilfe einer oder mehrerer Kameras aufgenommen, die auf der Brille befestigt sind. Dieses Kamerabild wird dem Benutzer in Echtzeit innerhalb des Displays angezeigt und mit virtuellen Informationen überlagert. Ein großer Vorteil dieses Displays ist, dass man das Kamerabild in Kontrast und in der Helligkeit kontrollieren und anpassen kann. Dadurch lässt sich beispielsweise Dunkelheit etwas ausgegleichen, was bei optischen See-Throughs nicht möglich ist. Aber auch das Video-See-Through hat verschiedene Nachteile. Durch die Aufnahme eines Kamerabildes gibt es Einschränkungen bezüglich der Video Bandbreite und der Auflösung. Das kann zu Verzögerungen bei der Bildabgleichung führen, wenn der Benutzer den Kopf bewegt. Im schlimmsten Fall führt es sogar zu Systemfehlern oder Blackouts [40, 50].

2.1.2 Virtual Reality (VR)

Die Virtual Reality (deutsch: virtuelle Realität) ist eine Technologie, die es dem Benutzer erlaubt, in einer computersimulierten Umgebung zu interagieren. Diese Umgebung versucht die Wirklichkeit in der Darstellung und in der Wahrnehmung zu simulieren und somit auch alle ihr zugrunde liegenden physikalischen Eigenschaften. Im Gegensatz zur AR taucht der Benutzer dabei komplett in die virtuelle Welt ein. Er soll in Echtzeit mit allen möglichen Gegenständen interagieren können und das Gefühl haben, Teil dieser virtuellen Welt zu sein. Das Gefühl, in eine virtuelle Welt einzutauchen, nennt man Immersion.



(a) Holobench [13]

(b) CAVE [50]

Abbildung 5: Arten von VR-Displays

Zur immersiven Darstellung virtueller Welten werden verschiedene Ausgabegeräte benötigt. Zu den einfacheren gehören Computermonitore. Des Weiteren gibt es Head-Mounted-Displays oder (Stereo-) Leinwände. Bei diesen Geräten nimmt der Benutzer allerdings die reale Umgebung um sich herum immer noch wahr. Daher versuchen andere Systeme die Realität immer mehr auszublenden. Dazu gehören Holobenches oder Caves (Abbildung 5(a) und 5(b)). Eine Holobench sind zwei aufeinander senkrecht stehende Leinwände, die zwei Projektoren in ihrem Gehäuse haben. Diese projizieren die computergenerierten Bilder von hinten auf die Leinwände. Der CAVE (Cave Automatic Virtual Environment) ist ein von Leinwänden umgebener Raum, an dessen Wände rückseitige Projektoren die Bilder werfen. Der Benutzer ist somit auch physisch eingeschlossen. Zur Positionserfassung von Objekten im Raum werden bei beiden Ausgabegeräten Tracking Systeme verwendet. Je weniger der Benutzer von der Realität wahrnehmen kann, desto stärker kann sein Gefühl der Immersion werden [50].

2.2 Echtzeitfähigkeit

Echtzeitfähigkeit ist eine Anforderung, die mittlerweile an viele meist industrielle Anwendungen gestellt wird. Ein System gilt als echtzeitfähig, wenn es unter allen Betriebsbedingungen richtig und rechtzeitig auf alle auftretenden Ereignisse reagiert. Ein Kommunikationssystem gilt – bezogen auf sein Anwendungsgebiet – als echtzeitfähig, wenn es die qualitativen und zeitlichen Forderungen an den Datenaustausch aller Komponenten erfüllt. Dies kann bedeuten, dass für die Zeit zwischen zwei Ereignissen eine bestimmte Maximalzeit nicht überschritten werden darf [11].

Die Echtzeitfähigkeit spielt in Mixed Reality-Umgebungen eine wichtige Rolle. Das folgende Szenario beschreibt ein echtzeitfähiges System, und was passiert, wenn die Echtzeitfähigkeit in diesem System nicht gewährleistet ist: Auf einer Messe werden neue VR-Systeme vorgestellt. Ein Besucher möchte das System testen. Dazu setzt er sich eine Datenbrille auf und es werden eventuell einige Marker an seinem Körper angebracht. Sobald er die Brille aufhat, findet er sich in einem virtuellen Flugzeug wieder. Dreht er seinen Kopf, so bewegt sich auch das Bild, so dass er in alle Richtungen im Flugzeug schauen kann. Bewegt er sich, so kann er sich im gesamten Raum bewegen und sich jeden Winkel aus der Nähe anschauen. Der Benutzer kann demnach komplett in diese virtuelle Welt eintauchen. Dieses Szenario beschreibt ein echtzeitfähiges System. Das Gefühl der Immersion bestätigt dies. Wäre die Simulation des virtuellen Flugzeugs nicht echtzeitfähig, so würde sich der Benutzer nicht wie in der echten Welt fühlen. Ein Ruckeln bei einer Kopfdrehung oder einer Bewegung könnte dies hervorrufen. Ruckeln bedeutet, dass das Bild, welches der Betrachter sieht, nicht so schnell berechnet wird, wie der Benutzer zum Fokussieren des neuen Bildes benötigt. Ein anderer Effekt könnte ein Nachziehen der Bilder sein. Der Benutzer sieht bei Kopfbewegungen nicht das aktuelle Bild, sondern vorherige. Das heißt, dass sich der Kopf schneller bewegt, als die Bilder berechnet werden, und diese sind daher zeitlich verschoben. Durch den Effekt des Ruckelns oder des Nachziehens wirken die Bilder auf den Benutzer nicht mehr flüssig. Ein solches System ist nicht echtzeitfähig.

Der menschlichen Wahrnehmung genügen zwischen 24 und 30 Bilder pro Sekunde [20], um eine „ruckelfreie“ Illusion der Kontinuität zu erzeugen, sofern sich die Einzelbilder nicht zu sehr von einander unterscheiden. Das VR-System sollte daher 25 Bilder pro Sekunde berechnen können, um echtzeitfähig zu sein.

Allerdings ist ein Schattenalgorithmus meist nur ein kleiner Teil eines VR-Systems. Verschiedene andere Algorithmen, wie z. B. die Darstellung der virtuellen Szene, oder das Tracking zur Positionserfassung von Objekten, sind wichtige Bestandteile dieses Systems. Außerdem werden für das Stereoempfinden des Benutzers meistens zwei Bilder innerhalb des gleichen Zeitraums generiert. Dies bedeutet, dass die Bilder mit allen benötigten Algorithmen mit mindestens 50 Frames pro Sekunde (fps) berechnet werden müssen, um ein Gefühl der Echtzeitfähigkeit zu vermitteln. Aus diesem Grund darf ein Schattenalgorithmus nur einen kleinen Anteil dieser Zeit beanspruchen.

2.3 Schatten

Als Schatten bezeichnet man das Gebiet hinter einem lichtdämpfenden oder opaken Gegenstand, welches komplett oder teilweise vom Licht verdeckt bleibt. Diese dunklere Fläche ist eine zweidimensionale Silhouette des geblockten Objektes, wobei die Größe und die Verzerrung vom Winkel der Fläche und vom Abstand zur Lichtquelle abhängen.

Im weiteren Verlauf der Arbeit werden Objekte, die den Weg des Lichtes zu einem Punkt versperren als *Blocker*, und Objekte, auf die ein Schatten fällt, als *Empfänger* bezeichnet.

2.3.1 Lichtquellen

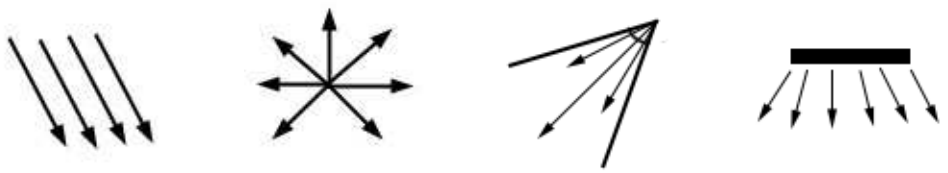
In der Natur existieren nur Lichtquellen, die eine gewisse räumliche Ausdehnung haben. Sie können also von einem Objekt auch nur teilweise bedeckt sein. Diese Art von Lichtquelle wird als *flächige Lichtquelle* bezeichnet. Die Form hängt dabei von der Art des Objektes ab. Um Licht in der Computergrafik zu simulieren, benutzt man die drei folgenden Arten von Lichtquellen:

Gerichtete Lichtquellen: Gerichtete Lichtquellen entsprechen einer unendlich weit entfernten Lichtquelle. In einer Szene, die von einer gerichteten Lichtquelle erhellt wird, kommt das Licht mit parallel einfallenden Strahlen aus einer Richtung.

Punktlichtquellen: Punktlichtquellen strahlen ihr Licht in alle Richtungen an ihre Umgebung aus. Ein gutes Beispiel dafür ist eine Glühbirne, dessen Licht keine spezielle Richtung hat.

Spotlichtquellen: Spotlichtquellen sind im Grunde Punktlichtquellen mit einer ganz bestimmten Richtung und einem Öffnungswinkel. Sie sind mit Scheinwerfern zu vergleichen, die auf einer Bühne eine ganz bestimmten Position ausleuchten.

Diese drei Arten von Lichtquellen reichen in vielen Situationen aus, um ein einfaches Licht in einer virtuellen Umgebung zu simulieren. Denn ab einer gewissen Entfernung ist es egal, ob der Betrachter eine flächige Lichtquelle betrachtet, oder ob sie durch eine Punktlichtquelle approximiert wird. Allerdings kann man auch mehrere Punktlichtquellen nebeneinander anordnen, um eine flächige Lichtquelle zu simulieren. Der Aufwand ist allerdings um ein Vielfaches höher, so dass es die Echtzeitfähigkeit meistens nicht zulässt. Daher gibt es in einer 3D-API wie OpenGL auch keine flächigen Lichtquellen, die man einfach an- oder ausschalten kann.



(a) gerichtetes Licht (b) Punktlichtquelle (c) Spotlichtquelle (d) flächige Lichtquelle

Abbildung 6: Arten von Lichtquellen

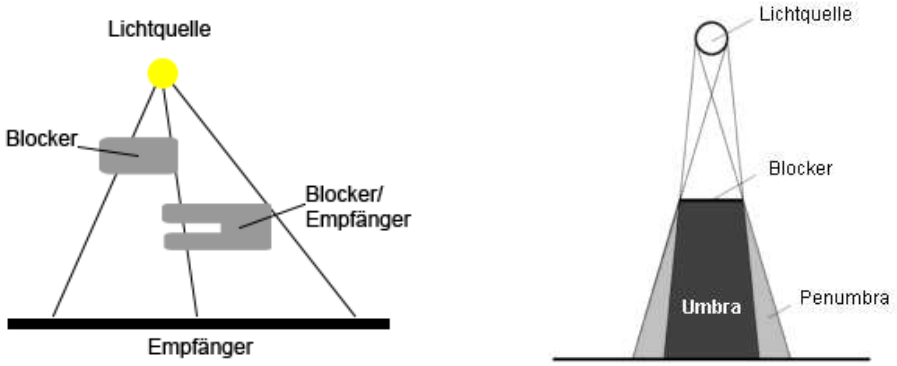
2.3.2 Schatten in der Natur

In der Natur unterscheidet man ganz allgemein zwischen den folgenden Schattenarten [14]:

Körperschatten Ein Körperschatten ist ein Schatten, der auf einem Körper selbst hervorgerufen wird. Der Körper selbst verdunkelt seine dem Licht abgewandte Seite. Der Blocker und der Empfänger sind demnach das selbe Objekt. Hier spricht man auch von Selbst-Verschattung.

Kernschatten (Umbra) Ein Kernschatten ist der dunkelste Bereich eines Schattens. Da in der Natur nur flächige Lichtquellen existieren, sind die Schatten nicht scharf begrenzt. Dies hat den Grund, dass Teile der Lichtquelle verdeckt sein können, andere Teile aber sichtbar sind. Ist die Lichtquelle klein genug bzw. weit genug entfernt, so existiert im Innern des Schattens ein Bereich, der von der Lichtquelle vollständig verdeckt wird. Dies ist der Kernschatten.

Halbschatten (Penumbra) Ein Halbschatten ist eine Fläche, die nicht das gesamte Umgebungslicht erhält. Er kann auf zwei Arten hervorgerufen werden: Durch eine flächige Lichtquelle (er entsteht dabei auf Flächen des Empfängers, die nur teilweise von der Lichtquelle verdeckt sind) oder durch die Existenz mehrerer Lichtquellen (dabei kann ein Kernschatten und um ihn herum maximal so viele Halbschattenflächen entstehen, wie Lichtquellen vorhanden sind).



(a) Selbstverschattung (b) Halbschatten [14]

Abbildung 7: Arten von Schatten

2.3.3 Schatten in der Computergrafik

In der Computergrafik versucht man, Schatten aus der Natur zu simulieren. Bei dieser Simulation können zwei verschiedene Schattenarten entstehen:

Harte Schatten Da die Lichtquellen aus der Natur, die flächigen Lichtquellen, schwer zu simulieren sind, benutzt man gerichtete, Punkt- und Spotlichtquellen[1]. Wird z. B. eine Punktlichtquelle angeschaltet, so kommt das Licht von einem einzigen Punkt aus. Liegt ein Objekt davor, dann verdeckt es die Lichtquelle entweder komplett oder gar nicht. Aus diesem Grund entsteht ein harter Schatten (*hard shadow*), welches nur dem Umbra-Teil des Schattens der Natur entspricht. Die Schattenkante des Schattens ist sehr hart, und die Farbe im Schatten ist an jeder Stelle gleich.

Weiche Schatten Weiche Schatten (*soft shadows*) haben im Gegensatz zum harten Schatten einen weichen Übergang an der Schattenkante. Von innen nach außen nimmt die Intensität der Farbe ab, und er wird weicher und verschwommener. Weiche Schatten sind eine gute Approximation der Schatten aus der Natur. Es bedarf aber meist einiger rechenintensiverer Algorithmen, als bei den harten Schatten, um sie zu berechnen.

(a) harter Schatten mit Shadow Volumes [21]

(b) weicher Schatten mit PPCF Shadow Mapping [9]

Abbildung 8: Schatten in der Computergrafik

[1] OpenGL unterstützt keine flächigen Lichtquellen

3 Bisherige Arbeiten

Da für den eigenen Ansatz die Echtzeitfähigkeit von Schatten der wesentlichste Punkt ist, wird sich dieses Kapitel hauptsächlich mit echtzeitfähigen Ansätzen für die Erzeugung von Schatten beschäftigen. Vor allem auf dem Gebiet der Soft Shadows existieren deutlich mehr, als die hier vorgestellten Ansätze. Da diese aber meist keine echtzeitfähigen Ergebnisse liefern, werden sie hier nicht aufgeführt. Eine gute Zusammenfassung hierzu bietet der STAR der Eurographics 2004 [6].

3.1 Planare Schatten

Die Idee der planaren Schatten von Jim Blinn [16] ist, dass diese als 2D-Projektion auf eine Ebene dargestellt werden. Dazu werden Geraden von der Lichtquelle durch die Eckpunkte der Objekte geschossen. Wenn man diese weiterverfolgt, findet man die Schatteneckpunkte in den Schnittpunkten von der Geraden mit der Ebene. Dazu erstellt man die Schattenprojektionsmatrix, die die schattenwerfenden Objekte auf eine zweidimensionale Ebene innerhalb der virtuellen 3D-Welt wirft und mit der Schattenfarbe rendert. Danach muss die Matrixmultiplikation wieder rückgängig gemacht und die Objekte mit der normalen Projektionsmatrix gerendert werden. Zur Bestimmung der Schattenprojektionsmatrix muss die Ebenengleichung der Ebene, auf die der Schatten geworfen werden soll und die Position des Lichtes bekannt sein. Das Hauptproblem hierbei ist allerdings, dass man meistens keine Schatten auf einer ganzen Ebene haben will. Dies kann zu einigen Clipping Fehlern führen, so dass z. B. ein Schatten auf einem Boden über den Boden hinaus in eine Wand hinein ragt.

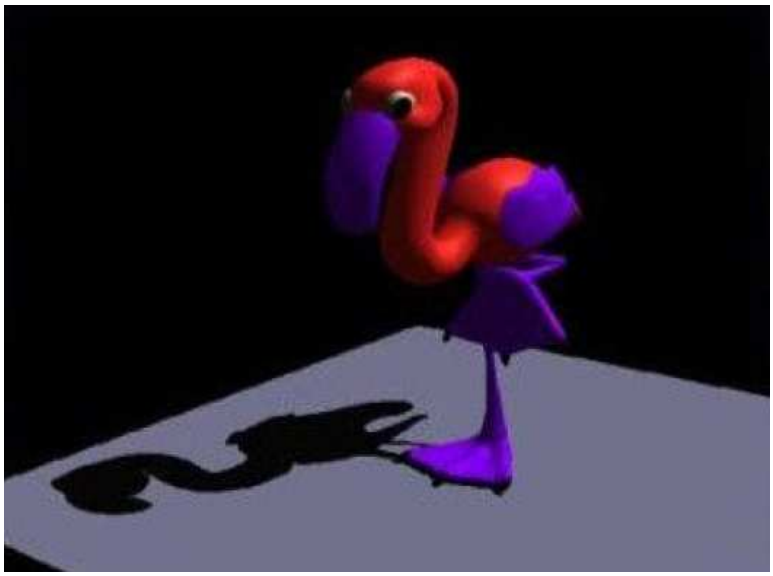


Abbildung 9: Planare Schatten [16]

3.2 Shadow Volumes

Der grundlegende Gedanke der Shadow Volumes (deutsch: Schattenvolumen) stammt von Franklin Crow aus dem Jahre 1977 [5]. Die Idee besteht darin, ein dreidimensionales Schattenvolumen hinter den schattenwerfenden Objekten zu berechnen, um somit testen zu können, ob andere Objekte im Schatten, also im Schattenvolumen oder außerhalb liegen. In der Hardware beschleunigten Variante des Original-Algorithmus [15] wird die Szene zunächst nur mit ambientem Licht aus Sicht der Kamera gerendert. Im zweiten Durchlauf wird das Shadow Volume in den Stencil Buffer gerendert, wobei die sichtbaren Pixel der front-facing Dreiecke den Stencil Buffer inkrementieren und die back-facing Dreiecke ihn dekrementieren. In dieser Schattenmaske im Stencil Buffer liegen jetzt alle Pixel im Schatten, deren Wert größer als Null ist. Im dritten Durchlauf wird die Szene mit kompletter Beleuchtung gerendert, wobei der Stencil Buffer dazu benutzt wird, die im Schatten liegenden Pixel auszumaskieren. Diese Methode heißt üblicherweise *Z-Pass* Shadow Volume Algorithmus.

Der große Vorteil von Shadow Volumes ist, dass sie sehr präzise und optimale Schatten erzeugen und keinen Aliasing-Effekt hervorrufen. Außerdem berechnen sie eine korrekte Selbst-Verschattung von Objekten. Nachteile sind, dass das Verfahren nur bei geschlossenen Meshes funktioniert. Des Weiteren muss ein Schattenvolumen von jedem schattenwerfenden Objekt berechnet werden, daher ist dieses Verfahren bei vielen Polygonen nicht sehr performant.

Abbildung 10: Schatten mit Shadow Volumes [3]

3.2.1 Verbesserungen

Der Z-Pass Algorithmus berechnet keinen korrekten Schatten, wenn die Near-Ebene der Kamera komplett oder teilweise innerhalb des Shadow Volumes liegt. Hier werden Teile des Shadow Volumes von der Near-Ebene der Kamera geclippt. Eine Verbesserung für diesen Fall heißt *ZP+* [44]. Die zugrunde liegende Idee ist es, den Stencil Buffer mit den Dreiecken vom schattenwerfenden Objekt, die innerhalb des Shadow Volumes liegen, vorzufüllen. Diese Dreiecke werden dann mit einer vorher konstruierten Projection Matrix auf die Near-Ebene der Kamera projiziert.

Eine andere Möglichkeit, Shadow Volumes zu rendern ist der *Z-Fail* [3] Algorithmus. In dieser Methode wird der Tiefentest beim Rendern des Shadow Volumes einfach umgekehrt, so dass nur die Dreiecke, die hinter dem Z-Buffer, also hinter den sichtbaren Oberflächen liegen, den Stencil Buffer verändern. Das Problem des Clippings mit der Near-Ebene ist damit gelöst, allerdings gibt es jetzt ein Problem mit dem Clipping an der Far-Ebene. Diese kann aber entweder ins Unendliche geschoben werden, oder die Tiefenwerte werden nicht geclippt sondern geclampt [3].

Verminderung der Rasterisierungsarbeit Es gibt Ansätze, die versuchen die Anzahl der Pixel, die beim Rendern des Shadow Volumes einbezogen werden, zu vermindern. Eine Methode zeigt, dass der Scissor Test zur Abgrenzung bei der Rasterisierung dienen kann, indem er den Lichtradius der Lichtquelle mit einbezieht [26]. NVIDIAs UltraShadow [34] erlaubt es den Programmierern, einen minimalen und einen maximalen Tiefenwert für den Shadow Volume zu definieren. Jetzt kann ein Pixel nicht im Shadow Volume sein, wenn der Z-Buffer Wert des Pixels außerhalb der definierten Tiefengrenzen liegt. Auf diese Art und Weise kann viel Rasterisierungsarbeit eingespart werden, wenn der Schattenempfänger außerhalb der Tiefengrenzen liegt. Als weitere Optimierung können die Minima und die Maxima auf weitere Shadow Volumes geclampt werden, wie z. B. die von Lichtquellen oder von einer Wand in einem Raum. Der Test mit den Tiefengrenzen ist allerdings nur dann wirklich effektiv, wenn das Shadow Volume ungefähr senkrecht zur Blickrichtung steht.

Die Kombination des Scissor Tests [26] und der Tiefengrenzen [34] mit einer neuen Optimierung, ist eine Umsetzung von McGuire [32]. Der eigentliche Schatten wird mit einer minimalen Anzahl von Dreiecken berechnet und das schattenwerfende Objekt wird der komplexeste Teil des Shadow Volumes. Da der Z-Pass Algorithmus das schattenwerfende Objekt nicht benötigt, ist es für Shadow Volumes, die sich mit dem Viewport überschneiden, schneller, den Z-Fail Algorithmus zu benutzen. Von Brabec und Seidel gibt es eine Methode, das Shadow Volume komplett auf der GPU zu berechnen [43]. Sowohl einige Stencil Shadow Optimierungen als auch das Rendern von Soft Shadows wird von Lengyel vorgestellt [27]. Mit so genannten *geometry scissors*, also der Betrachtung der Geometrie des Schattenempfängers, kann etwas Shadow Volume Rendering Arbeit verringert werden. Lloyds Methode [2] clampt Shadow Volumes, so dass die Rasterisierung nur auf Regionen begrenzt ist, die Schatten-Empfänger beeinhaltet.

Die Idee hinter der Methode von Laine [45] ist es, lokal zwischen dem Z-Pass und dem Z-Fail Algorithmus zu wählen. Es wird gewählt zwischen einem niedrig aufgelösten Tiefenpuffer und einer automatisch konstruierten Split Ebene. So wird die Anzahl der Stencil Updates verringert, ohne den resultierenden Schatten zu beeinträchtigen. Von Aila und Akenine-Möller stammt eine hierarchische Methode, Shadow Volumes auf der GPU zu rendern [52]. Zunächst wird ein Shadow Volume mit kleiner Auflösung rasterisiert und Pixelblöcke, die eine Shadow Volume Grenze beeinhalten könnten, werden detektiert. Als zweites wird das Shadow Volume auf Pixelbasis innerhalb der Grenzen rasterisiert.

Hybride Algorithmen Chan und Durand [8] kombinieren die beiden wichtigsten Schatten-Algorithmen Shadow Mapping und Shadow Volumes. Zuerst werden durch eine niedrig aufgelöste Shadow Map die Randpixel der Schattenregionen herausgefunden. Mithilfe dieser Randpixel wird das Shadow Volume erstellt, und die übrigen Pixel der Szene werden mit der Shadow Map behandelt.

Soft Shadows Einige Erweiterungen des Shadow Volume-Algorithmus, so wie die Penumbra Wedges [55, 56], erzeugen einen qualitativ guten Soft Shadow in Echtzeit, allerdings nur bei nicht allzu komplexen Szenen. Hierbei wird von jeder Ecke der Silhouette des Schattenwerfers eine *penumbra wedge* extrudiert, die den Halbschatten darstellt.

Abbildung 11: Weicher Schatten [21]

3.3 Shadow Mapping

Auch der Grundgedanke des Shadow Mapping stammt schon aus dem Jahre 1978 von Lance Williams [25]. Die Idee besteht darin, die Tiefenwerte aus Sicht der Lichtquelle in einer Textur, der sogenannten Shadow Map, zu speichern. Mit dem Vergleich des echten Tiefenwertes und dem aus der Shadow Map wird ermittelt, ob das jeweilige Pixel im Schatten liegt oder nicht.

Zunächst wird die Szene von der Lichtquelle aus gerendert. Der Z-Buffer wird ausgelesen und in der Shadow Map gespeichert. Die Shadow Map ist eine 2D-Textur, die nur aus Grauwerten besteht. Je dunkler ein Pixel in der Shadow Map, desto näher ist es zur Lichtquelle. Jetzt wird die Szene ein zweites Mal, nämlich aus Sicht der Kamera gerendert. Die Shadow Map wird nun von der Lichtquelle aus auf die Szene projiziert. Hierbei wird pro Punkt der eigentliche Z-Wert mit dem Tiefenwert aus der Shadow Map verglichen. Sind diese Werte gleich, so befindet sich kein Objekt auf der Sichtlinie zur Lichtquelle. Daher kann normal beleuchtet werden. Ist allerdings der eigentliche Z-Wert, d.h. der Abstand des Punktes zur Lichtquelle größer als der Wert in der Shadow Map, liegt noch ein Objekt zwischen dem Punkt und der Lichtquelle. Also liegt er im Schatten.

Der Vorteil von Shadow Mapping besteht darin, dass die Komplexität der Szene beliebig ist. Daher ist das Verfahren vor allem bei grossen Modellen schneller als Shadow Volumes. Zusätzlich unterstützt es die Selbstverschattung von Objekten. Außerdem ist die Implementation komplett auf der GPU möglich, wodurch wieder ein Geschwindigkeitsvorteil erzielt wird. Nachteile sind eckige Schattenkanten, der so genannte Aliasing-Effekt. Außerdem funktioniert das Verfahren nur mit Spotlights, nicht mit Punktlichtquellen. Für die meisten Nachteile wurden allerdings schon einige Verbesserungen vorgestellt, die zum Teil in Kapitel 3.3.1 nachzulesen sind. Abbildung 12 zeigt einen Ausschnitt aus Pixars „Toy Story“. Dieser Film wurde komplett mit der Software *Renderman* gerendert, in der der Shadow Map Algorithmus implementiert ist.

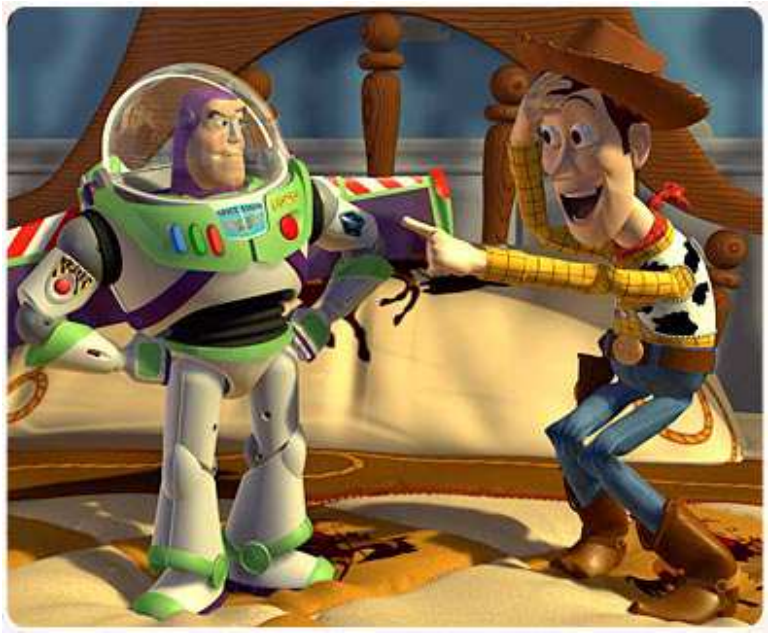


Abbildung 12: Ausschnitt aus Pixars „Toy Story“ [38]

3.3.1 Verbesserungen

Da das Hauptproblem des Shadow Mapping das Aliasing ist, beschäftigen sich viele Ansätze mit dem Anti-Aliasing. Es existieren aber auch sehr viele Möglichkeiten für Soft Shadows mithilfe von Shadow Maps.

Anti-Aliasing Perspective Shadow Mapping [28] versucht, die Aliasing-Effekte durch eine perspektivische Transformation der Szene vor Erstellung der Shadow Map zu reduzieren. So steht Objekten, die näher an der Kamera liegen, eine höhere Auflösung bei der Erzeugung der Shadow Map zur Verfügung, als denen, die weiter weg in der Szene liegen. Der größte Nachteil des Verfahrens besteht darin, dass nahe Objekte sehr groß erscheinen, wogegen die Größe schon nach kurzer Blickweite erheblich abnimmt. Das führt zu einem schnellen Qualitätsverlust nach kurzen Distanzen, so dass die Aliasing-Effekte von hinteren Objekten stärker sein können als beim Standart Shadow Mapping. Diesen Nachteil versuchen die Light Space Perspective Shadow Maps [31] zu vermindern, indem sie durch eine perspektivische Transformation alle Lichter als direktionale Lichter behandeln kann, und somit nicht die Richtung der Lichtquellen verändern müssen. Auch der Ansatz des Trapezoidal Shadow Mappings [53] baut auf dem Perspective Shadow Mapping auf, benutzt allerdings ein zum Einheitswürfel verzerrtes Trapez, um den perspektivischen Effekt zu erzielen. Andere Verfahren, wie die Shadow Silhouette Maps [39] fügen Punkte der Silhouette zur Shadow Map hinzu, um die Präzision zu erhöhen und somit den Aliasing-Effekt zu verringern. Das PCF (Percentage Closer Filtering) Shadow Mapping [58], reduziert den Aliasing-Effekt, indem er verschiedene Tiefenvergleiche der umliegenden Texturpixel durchführt und diese Ergebnisse bilinear interpoliert. Dieses Verfahren kann außerdem verwendet werden, um Soft Shadows zu approximieren. Des Weiteren gibt es dazu eine Verbesserung [9], die abhängig von der Höhe des Schattens weicher oder nicht zeichnet.

Soft Shadows Eine weitere Methode, die nur ein einziges Licht-Sample zur Erstellung von Soft Shadows benötigt [51], und von Brabec für die GPU verändert wurde [49], schafft es meist nur kleine Penumbra-Regionen in Echtzeit zu erstellen. Die Erweiterung von Kirsch [10] ermöglicht eine schnelle Möglichkeit der Erstellung von Soft Shadows, allerdings mit einigen Artefakten. Penumbra Maps [4] haben eine große Ähnlichkeit mit Penumbra Wedges. Sie berechnen zunächst eine Shadow Map aus einer Lichtquelle heraus. Im zweiten Teil berechnen sie eine Penumbra Map, mit dessen Intensität und der Tiefeninformation aus der Shadow Map sie im dritten Durchlauf die Szene rendern. Sie erreichen eine gute Qualität vor allem bei runden Lichtquellen, aber erzeugen einige Artefakte bei zunehmender Größe der Lichtquelle. Von der Qualität vergleichbar mit den Penumbra Maps, aber ein bisschen schneller, sind Smoothies [7]. Smoothies sind 2D Primitive, die aus der Verlängerung der Silhouettenkanten des schattenwerfenden Objektes entstehen und die Penumbra-Region simulieren. Schönere Penumbra-Regionen erstellt der Ansatz von de Boer [57]. Er kann sogar nicht-polygonale Objekte rendern, hat aber einige Performanceschwächen bei großen Schatten. Sein Algorithmus verwendet so genannte *skirts*. Skirts sind Datenstrukturen, die die Tiefeninformationen und die Informationen über die Schattenabschwächung eines schattenwerfenden Objektes beeinhalten.

Eine andere Erweiterung der Shadow Maps, die genau einen einzelnen Tiefenwert pro Pixel abspeichert, sind die Deep Shadow Maps [54]. Diese speichern eine Repräsentation der teilweisen Sichtbarkeit eines Pixels in allen möglichen Tiefen ab. Gute Ergebnisse in Echtzeit erzielt eine etwas neuere Erweiterung der Deep Shadow Maps, die Penumbra Deep Shadow Maps (PDSM) [18]. Diese erlauben es, neue Objekte in die Szene hinzuzu-

fügen, ohne die PDSM neu generieren zu müssen.

Abbildung 13: Inhalt einer Shadow Map: die dunkleren Pixel liegen näher an der Lichtquelle und die helleren sind weiter entfernt

3.4 Ansätze für Schatten in AR-Umgebungen

Es gibt verschiedene Ansätze, Schatten auch für die Augmented Reality zu generieren. Meist ist es allerdings ein Verfahren, dass sich zusammensetzt aus einem der beiden Schattenverfahren Shadow Mapping oder Shadow Volumes, und verschiedenen anderen Techniken, zum Teil auch aus der Bildverarbeitung. Das grundlegende Vorgehen ist dabei meistens folgendes: Zuerst müssen die realen Lichtquellen im Bild gefunden werden. Dazu werden meistens erst die Schatten detektiert, um auf die Lichtquellen schließen zu können. Erst danach kann man die virtuellen Schatten in der realen Szene rendern. Zusammengefasst sind einige Verfahren im STAR der Eurographics 2004 [22]. Voraussetzung für diese Verfahren ist jeweils ein virtuelles Modell der realen Szene und mindestens ein Bild davon.

Eine frühe Arbeit aus dem Jahre 1994 [1] platziert schon virtuelle Objekte in einer see-through-realen Szene. Eine reale Lichtquelle wird getrackt und bewegt, so dass die virtuellen Objekte in der Szene Schatten auf die reale Szene werfen. Diese Arbeit verwendet den Shadow Mapping-Algorithmus.

Ein guter Ansatz stammt aus dem Jahre 2003 [30] und stellt einen modifizierten Echtzeit Shadow Volume-Algorithmus vor, der für die Augmented Reality genutzt werden kann. Man benötigt die Position und Richtung der realen Lichtquelle, sowie die Phantomobjekte der realen Objekte, um korrekte Schatten der virtuellen Objekte darstellen zu können. Der Algorithmus rendert die Szene zwei Mal. Beim ersten Durchlauf werden alle realen Objekte, inklusive der Schatten für die virtuellen Objekte gerendert. Beim zweiten Durchlauf werden nur die virtuellen Objekte gerendert. Hierbei wird für alle schattenwerfenden Objekte ein Shadow Volume generiert, für dessen Berechung man die Phantomobjekte benötigt. Das Problem, das hierbei auftritt ist, dass die Überschneidung verschiedener Schatten nicht korrekt dargestellt wird.

Eine weitere Methode [23], die ebenfalls Shadow Volumes zum finalen Rendern der virtuellen Schatten verwendet, aber auch Shadow Mapping zulassen würde, hat drei Rendering-Schritte. Im ersten Schritt, der Schatten Erkennung, werden die realen Schatten in der Szene anhand von Textur Informationen und Estimationen der Schatten-Regionen gefunden. Der nächste Schritt erstellt eine *protection-mask*, die die Schatten-Regionen für weiteres Rendern schützt. Am Ende werden die virtuellen Schatten mittels Shadow Volumes gerendert und zur Szene hinzugefügt. Hierbei wird die Überschneidung mehrerer Schatten korrekt berechnet. Außerdem braucht man nur eine ungefähre Position der Lichtquelle und eine ungefähre Geometrie des Phantomobjektes.

Gibson [47, 48] entwickelte zwei weitere Methoden, um Soft Shadows in einer Mixed Reality-Umgebung zu simulieren. Allerdings benötigt man eine sehr genaue Geometrie der Phantomobjekte und eine gute Schätzung der BRDF, also des virtuellen Beleuchtungsmodells und der originalen Beleuchtung. Aus den Beispielen der Paper geht allerdings nicht eindeutig hervor, ob die Überschneidung von Schatten korrekt behandelt wird.

4 Schattenverfahren im Detail

4.1 Shadow Mapping

Wie es in Kapitel 3.3 schon grob skizziert wurde, sind die Schritte, um einen Schatten mithilfe von Shadow Maps zu erzeugen folgende:

1. Render die Szene aus Sicht der Lichtquelle
2. Speichere den Tiefenwert in einer Textur (Shadow Map)
3. Projiziere die Shadow Map-Textur auf die Szene
4. Benutze das Ergebnis des Visibilitätstests für die Farbe im Fragment Shader

Im dritten Schritt werden dazu projektive Texturen benötigt. Der folgende Abschnitt beschäftigt sich damit.

4.1.1 Projektive Texturen

Texturieren bedeutet normalerweise, dass man explizit Texturkoordinaten festlegt, um eine Textur auf eine Oberfläche zu legen.

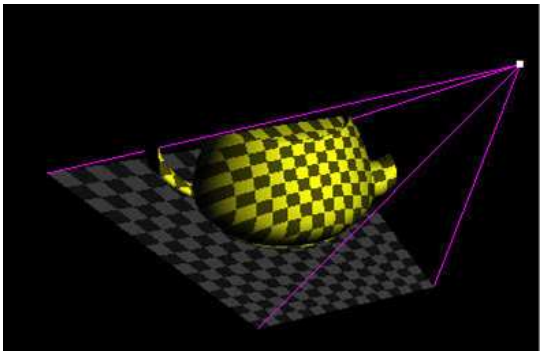


Abbildung 14: Projektive Textur

Bei projektiven Texturen projiziert man die Texturen auf eine Oberfläche, also wie von einem Beamer. Verschiebt man die Oberfläche, so verändert sich auch ihre Textur, da sie nun auf andere Texturkoordinaten zugreift. Abbildung 14 zeigt, wie eine projektive Textur auf eine Szene gelegt wird. Das Prinzip ist, dass man durch eine Sequenz von Transformationen, Objekt-Koordinaten in 2D-Koordinaten, also Texturkoordinaten verschiebt, und dadurch herausfindet, an welcher Stelle jeder Eckpunkt auf die Textur passt. Danach benutzt man die gefundene Position als Texturkoordinate für den Eckpunkt, der beim Rendern die passende Textur auf das jeweilige Dreieck legt.

Da OpenGL im Grunde nur planare Projektionen für die Texturkoordinaten durchführen kann, wird die Texturmatrix für eine Projektion benutzt.

$$T = S * P_L * M_L * M_C^{-1} \tag{1}$$

S aus Formel 1 ist die Bias Matrix:

$$S = \begin{pmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \tag{2}$$

Formel 1 zeigt die Sequenz von Transformationen, die für eine projektive Texturierung angewendet werden muss. Als erste Transformation kommt die inverse Kamera-Modelviewmatrix M_C^{-1}. Sie wird benötigt, da die Textur von der Lichtposition auf die Welt projiziert werden soll. Daher muss zunächst die Transformation der Kamera aufgehoben werden. Danach kommt M_L, die Modelviewmatrix des Lichtes. Durch sie wird ein Punkt so rotiert und transliert, so dass er sich danach in der Lichtquelle befindet. Nun kommt die Transformation durch die Projektionsmatrix des Lichtes P_L, die das Licht Frustrum definiert.

Mithilfe dieser drei Multiplikationen befinden sich die x-, y- und z-Werte des transformierten Eckpunktes zwischen -1 und 1 und in der Position der Lichtquelle. Da Texturen in dem Wertebereich von 0 bis 1 liegen müssen, ist es notwendig sie in diesen zu verschieben. Dies geschieht mit der Multiplikation der x, y und z Komponenten des Punktes mit $\frac{1}{2}$ und der Addition nochmals mit $\frac{1}{2}$. Diese Transformation wird durch die Bias Matrix *S* (Formel 2) gemacht.

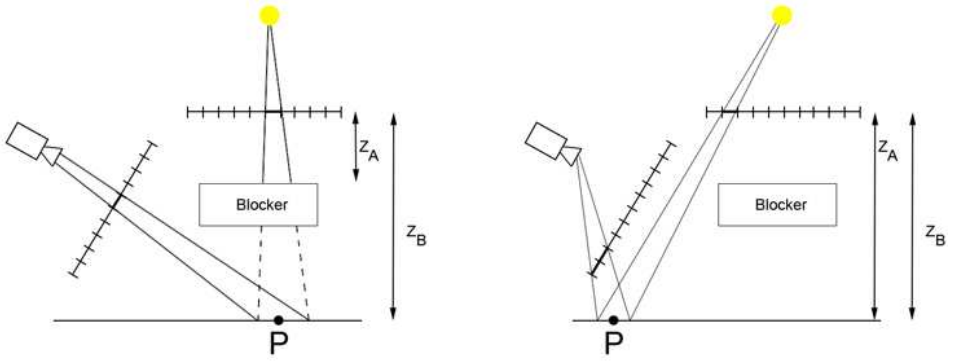
4.1.2 Tiefentest

Mithilfe der projektiven Texturen wird der dritte Schritt des Shadow Mappings ausgeführt. Doch zunächst müssen die ersten beiden Schritte ausgeführt werden. Dazu wird die Szene zuerst von der Lichtquelle aus gerendert. Die Kamera wird hierzu mit einer Modelviewmatrix in die Lichtquelle transformiert. Außerdem wird die Projektionsmatrix auf die des Lichtes gesetzt. Beim Rendern wird nur der Tiefenwert jedes einzelnen Pixels ausgelesen und in der Tiefen-Textur, der so genannten Shadow Map gespeichert. Ein Beispiel hierzu wird in Abbildung 13 gezeigt. In diesem Graustufenbild sind die dunkleren Pixel Punkte, die näher am Licht sind und die helleren Pixel Punkte, die sich weiter vom Licht entfernt befinden.

Im nächsten Schritt wird die Szene aus Sicht der Kamera gerendert. Dabei wird das Verfahren der projektiven Textur angewendet. Denn während dieses Renderprozesses, wird jeder Punkt von Kamerakoordinaten in Lichtkoordinaten transformiert. Dieser transformierte Tiefenwert Z_B wird dann mit dem korrespondierenden Wert Z_A aus der Shadow Map verglichen. Dieser Vergleich wird in Abbildung 15 verdeutlicht. Die Lichtquelle schaut durch die Shadow Map auf den Punkt P (Z_A) und die Kamera durch die Pixel im darzustellenden Bild (Z_B). Wenn der transformierte Tiefenwert größer ist als der Wert aus der Shadow Map, so liegt ein Objekt dichter am Licht, und der Punkt liegt dadurch im Schatten. Auf der anderen Seite liegt ein Punkt im Licht, wenn der transformierte Tiefenwert und der Wert aus der Shadow Map gleich groß sind, da es kein Objekt blockt (Formel 3).

$$\begin{aligned} Z_B > Z_A &\Rightarrow \text{Punkt ist im Schatten} \\ Z_B = Z_A &\Rightarrow \text{Punkt wird beleuchtet} \end{aligned} \tag{3}$$

Theoretisch ist Z_A also niemals kleiner als Z_B, da Z_B der Abstand zu dem Objekt ist, dass am nächsten zum Licht liegt.



(a) $Z_B > Z_A$ (b) $Z_A = Z_B$

Abbildung 15: Der Tiefentest

Mit dem Shadow Mapping Algorithmus lässt sich ein korrekter Schatten in Echtzeit erstellen. Die Geschwindigkeit ist dabei nicht von der Komplexität der Szene abhängig, wodurch er schneller als der Ansatz der Shadow Volumes ist. Allerdings hat er den großen Nachteil, dass durch die endliche Auflösung der Shadow Map am Schattenrand der sogenannte

Aliasing-Effekt auftritt. Dieser Effekt kann z.B. durch das Percentage Closer Filtering, welches im folgenden Unterkapitel vorgestellt wird, verringert werden.

4.2 PCF-Shadow Mapping

Benutzt man den vorgestellten Standard Shadow Mapping-Algorithmus, so führt dies zu zwei unschönen Ergebnissen. Ein generelles Problem des Shadow Mappings ist der so genannte *Aliasing-Effekt*. Aliasing ist ein Treppeneffekt, der an den Schattenrändern auftritt. Betrachtet man die Ränder des erstellten Schattens genauer, so stellt man fest, dass sie gezackt sind. Abbildung 16 verdeutlicht den Aliasing-Effekt. Links im Bild ist eine mit Standard-Shadow Mapping gerenderte Szene. In dem Ausschnitt auf der rechten Seite des Bildes sieht man eine sehr gezackte und damit unschöne Schattenkante.

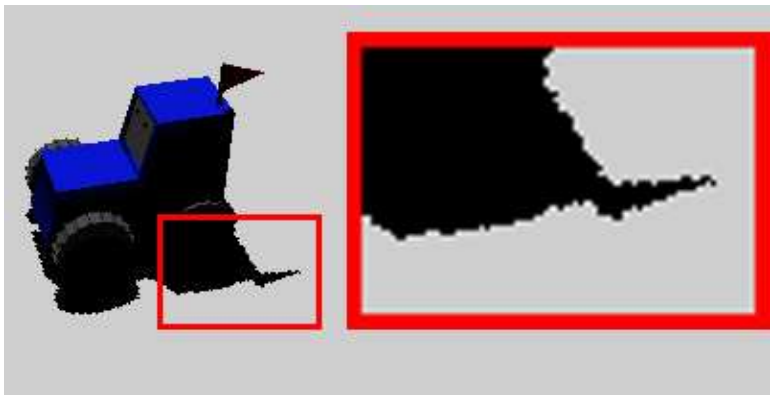


Abbildung 16: Aliasing-Effekt

Dieser Effekt entsteht durch die endliche Auflösung der Shadow Map. Beim Schattentest werden verschiedene Pixel der Szene auf den selben Punkt der Shadow Map transformiert und daher mit dem gleichen Wert verglichen. Es entstehen quadratische Schattenbereiche, die den Rand des Schattens unschön aussehen lassen. Dieses Problem könnte man mit dem Einsatz einer höher aufgelösten Shadow Map verringern. Allerdings ist diese Lösung nicht sehr praktikabel, da bei jeder größeren Shadow Map der Rechenaufwand steigt. Hat man jetzt eine sehr polygonreiche Szene, so wird der Algorithmus zu langsam werden.

Das zweite Problem des Standard Shadow Mapping-Algorithmus ist, dass er nur den Schatten von Spotlichtquellen darstellen kann. Schatten von flächigen Lichtquellen, also Lichtquellen, die eine weiche Schattenkante erzeugen, können mit diesem Algorithmus nicht simuliert werden.

Um diese Probleme in Ansätzen zu beheben, dient das Percentage Closer Filter-Shadow-Mapping, kurz PCF-Shadow-Mapping. Es behebt also nicht nur das Aliasing-Problem, sondern erzeugt auch weiche Schattenkanten, nämlich uniforme Soft-Shadows. Das bedeutet, dass die Schattenkanten über einen konstanten Abstand verschwimmen und dadurch das Aliasing verstecken.

Der Unterschied zum Standard Shadow Mapping besteht an der Anzahl der ausgeführten Schattentest pro Pixel. Standard Shadow Mapping führt pro Pixel einen Schattentest aus und das PCF-Shadow Mapping mehrere. Das aktuelle Pixel wird hierzu zusätzlich mit den umliegenden Tiefenwerten aus der Shadow Map verglichen. Diese Ergebnisse werden addiert um daraus den Durchschnittswert zu ermitteln, der als Schattenwert für das Pixel benutzt wird.

In der oberen Reihe in Abbildung 17 sieht man einen Schattentest, wie er beim Standard Shadow Mapping gemacht wird. Der Mittelpunkt des Pixels ist durch den roten Punkt

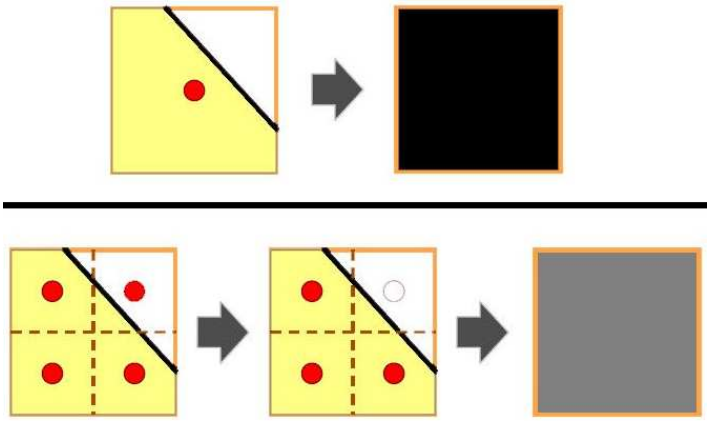


Abbildung 17: Prinzip eines PCF-Filters [41]

gekennzeichnet. Der Tiefenwert des roten Punktes wird für den Schattentest des jeweiligen Pixels verwendet, also mit dem Wert aus der Shadow Map verglichen. Da im oben gezeigten Beispiel der Mittelpunkt des Pixels von schattierter Fläche überzogen ist, den der Schattentest ermittelt hat, so wird das gesamte Pixel als schattiert angesehen, also schwarz gezeichnet.

Die untere Reihe der Abbildung zeigt die Funktionsweise eines 4-Sample-PCF-Filters. Die Anzahl der hier gewählten Samples ist variabel. Je mehr man zur Berechnung des Schattens benutzt, desto verschwommener sehen die Schattenkanten aus und um so größer wird der Bereich der Penumbra und die gesamte simulierte Fläche der Lichtquelle. Ein Pixel beim 4-Sample-PCF-Filter wird um drei Bereiche erweitert, wobei zu jedem Mittelpunkt der Flächen ein eigener Schattentest durchgeführt wird. Im Beispiel entsteht eine dunkelgraue Pixelfarbe, da drei der vier durchgeführten Tests eine Schattierung zurückliefern und einer eine beleuchtete Fläche. Demnach sind 75 % des Pixels schattiert, welches zu einer dunkelgrauen Pixelfarbe führt.

PCF-Shadow Mapping beginnt wie der Standard Algorithmus mit dem Rendern der gesamten Szene aus der Sicht der Lichtquelle. Hierbei speichert er wieder die Tiefenwerte in der Shadow Map. Der Unterschied der Algorithmen liegt in der Anzahl der durchgeführten Schattentests. Standard Shadow Mapping führt einen einzigen Schattentest mit dem Wert aus der Shadow Map pro Pixels aus, so dass das Pixel zwei Werte annehmen kann: schattiert oder beleuchtet. Um weiche Schattenkanten darstellen zu können, reichen diese beiden Zustände nicht aus. Aus diesem Grund bezieht das PCF-Shadow-Mapping die umliegenden Tiefenwerte aus der Shadow Map in die Berechnung mit ein. Je mehr zusätzliche Tiefenwerte mit einbezogen werden, desto mehr Schattentests müssen letztendlich ausgeführt werden. Ein Zähler wird jedes Mal inkrementiert, wenn ein Schattentest eine schattierte Fläche zurückliefert. Am Ende wird der Durchschnittswert ermittelt, indem man den Zähler durch die Gesamtanzahl der durchgeführten Schattentests dividiert. Dieser Durchschnittswert ist der Schattenfaktor des betrachteten Pixels und gibt an, wie stark das Pixel schattiert ist. Je mehr Schattentests fehlschlagen, desto kleiner ist der Durchschnittswert und desto heller wird das Pixel letztendlich schattiert.

Ein PCF-Filter läßt die Schattenkanten über einen konstanten Abstand verschwimmen, welches den Aliasing Effekt verdeckt. Dabei entsteht der Eindruck eines weichen Schattens. Die Weichheit hängt dabei von der Anzahl der verwendeten Tiefentests ab. Allerdings

werden dadurch keine physikalisch korrekten Soft-Shadows erzeugt, da sich die Schattenkante nicht in Abhängigkeit von der Größe der Lichtquelle und dem Abstand zum Empfänger verändert. Allerdings kann mit dem Verfahren der Eindruck eines weichen Schattens erzeugt werden, ohne den Shadow Mapping Algorithmus entscheidend zu verlangsamen.

4.3 PCSS-Shadow Mapping

Basierend auf dem Standard Shadow Mapping und dem Percentage Closer Filtering (PCF) existiert der Ansatz der Percentage Closer Soft Shadows (PCSS Shadow Mapping). Der Vorteil zu normalem PCF Shadow Mapping ist, dass PCSS korrekte perspektivische Soft-Shadows erzeugt, so genannte Schlagschatten. Dies sind weiche Schatten, die in Abhängigkeit der Größe der Lichtquelle und dem Abstand des Empfängers verschieden weit verlaufen und so deutlich realistischer wirken.

Dazu wird die Anzahl der Nachbarpixel, mit denen der Schattentest durchgeführt wird, nicht von vornherein festgelegt, sondern im Algorithmus gewählt. Aus diesem Grund ändern sich die Größe der Umbra und der Penumbra abhängig von der Größe der Lichtquelle, dem Abstand des Blockers und dem Abstand des Empfängers zur Lichtquelle. Der Algorithmus führt dazu die drei folgenden grundlegenden Schritte aus:

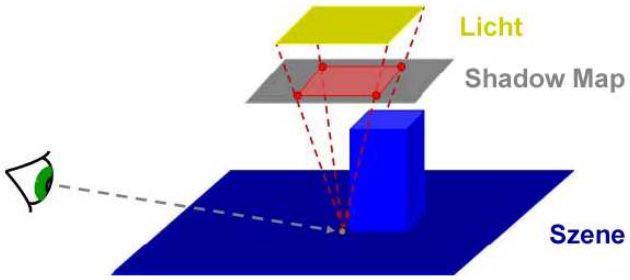


Abbildung 18: Bestimmung einer Suchregion auf der Shadow Map [41]

Schritt 1: Suche nach Blockern Hier wird auf der Shadow Map nach allen Tiefenwerten gesucht, die näher an der Lichtquelle als am Empfänger sind. Von diesen Werten wird der Durchschnitt ermittelt. Das Ergebnis ist ein durchschnittlicher Tiefenwert der gefundenen Blocker, der für die Kalkulation der Penumbra wichtig ist. Um nicht immer auf der gesamten Shadow Map suchen zu müssen, wird eine Suchregion bestimmt. Die Größe der Suchregion auf der Shadow Map hängt dabei von der Größe der Lichtquelle und von der Distanz zwischen dem Empfänger und der Lichtquelle ab. Abbildung 18 zeigt eine solche Suchregion auf der Shadow Map.

Schritt 2: Kalkulation der Penumbra Die Breite der Penumbra wird in Formel 4 basierend auf der Größe der Lichtquelle und den Distanzen vom Blocker und vom Empfänger zum Licht geschätzt. Dazu wird eine Approximation von zwei parallelen Ebenen benutzt, d.h. die Formel basiert auf der Annahme, dass Lichtquelle, Blocker und Empfänger parallel zueinander sind.

$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) * w_{Light}}{d_{Blocker}} \tag{4}$$

Die zu ermittelnde Größe der Penumbra ist $w_{Penumbra}$. Dafür benötigt man die Abstände des Empfänger- und Blockerpixels zur Lichtquelle $d_{Receiver}$ und $d_{Blocker}$ und die Größe der Lichtquelle w_{Light}. Abbildung 19 verdeutlicht dies.

Schritt 3: Filterung Im letzten Schritt wird ein Percentage Closer Filtering, wie es in Kapitel 4.2 vorgestellt wurde, angewendet. Allerdings basiert es nicht mehr auf einer

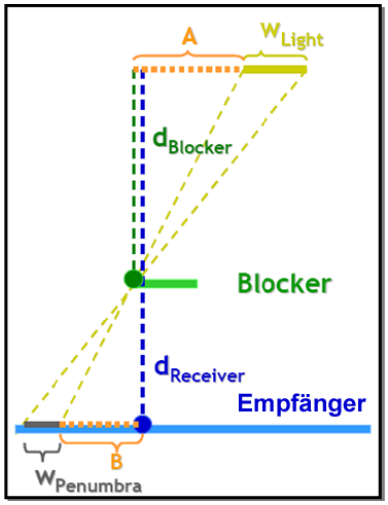


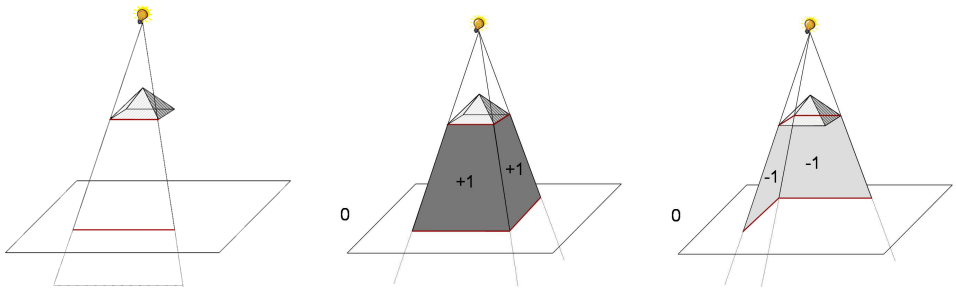
Abbildung 19: Kalkulation der Penumbra [41]

vorher festgelegten Filtergröße. Die Filtergröße ist die Variable $w_{Penumbra}$ aus Schritt 2, die für jedes Pixel unterschiedlich groß sein kann. Daher nennt man dieses Verfahren auch variablen PCF-Filter.

Der Ansatz der PCSS erzeugt eine weiche Schattenkante, die in Abhängigkeit von der Größe der Lichtquelle und von dem Abstand des Empfängers verschieden weit verläuft. Dadurch wirkt sie im Vergleich zu einem normalen PCF-Filter realistischer. Allerdings ist ihre Berechnung deutlich aufwändiger, so dass sie die Geschwindigkeit des Shadow Mapping Algorithmus deutlich verlangsamt.

4.4 Shadow Volumes

Der Shadow Volume-Algorithmus projiziert einen Strahl von der Lichtquelle aus durch jeden Eckpunkt des Blockers, bis hin zu einem Punkt, der normalerweise im Unendlichen liegt. Diese Projektionen zusammen formen das Schattenvolumen. Alle Punkte innerhalb dieses Schattenvolumens sind im Schatten und alle außerhalb werden durch die Lichtquelle beleuchtet. Da sich innerhalb einer Szene mehrere Objekte und dadurch auch mehrere Schattenvolumen befinden, so findet ein Schattentest heraus, welches Pixel schattiert wird und welches nicht. Dazu werden pro Pixel die Schnittpunkte aus Sicht des Betrachters der Schattenvolumen gezählt. Tritt man in ein Volumen ein, so zählt man +1, tritt man wieder heraus, so zählt man -1. Ist das Ergebnis dieses Tests größer als null, so befindet sich das Pixel innerhalb eines Schattenvolumens und daher im Schatten. Dieser Test wird normalerweise im Stencil Puffer implementiert.



(a) (b) (c)

Abbildung 20: Erstellung eines Schattenvolumens [50]

Um ein Schattenvolumen zu erstellen sind folgende Schritte notwendig:

Schritt 1: Geometrie zeichnen Die gesamte Szene wird als erstes gerendert und die Silhouettenkanten werden verlängert. Siehe Abbildung 20(a).

Schritt 2: Stencil Puffer mit 0 löschen Um Werte in ihm speichern zu können, muss der Stencil Puffer erst mit 0 initialisiert werden.

Schritt 3: Farbpuffer abschalten Da die Schattenvolumen nicht angezeigt werden sollen, wird der Farbpuffer abgeschaltet.

Schritt 4: Tiefenpuffer auf read-only stellen Dieser Schritt ist notwendig, damit die Rückseiten, die meistens von den Vorderseiten verdeckt sind, im Tiefenpuffer nicht überschrieben werden.

Schritt 5: Stencil Puffer einstellen Beim Zeichnen eines Polygons müssen die sichtbaren Bereiche um eins erhöht werden.

Schritt 6: Backface Culling aktivieren So werden die Rückseiten eines Objektes nicht gezeichnet.

Schritt 7: Schattenvolumen zeichnen Durch Backface Culling werden nur die Vorderseiten des Schattenvolumens gezeichnet und der Stencil Buffer an diesen Stellen um eins erhöht. Abbildung 20(b) zeigt das Ergebnis.

Schritt 8: Stencil Puffer einstellen Jetzt müssen beim Zeichnen eines Polygons die sichtbaren Bereiche um eins erniedrigt werden.

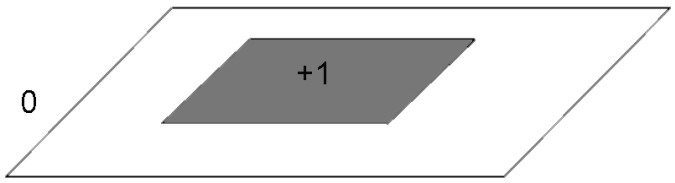


Abbildung 21: Der entstandene Schatten mit Shadow Volumes [50]

Schritt 9: Frontface Culling aktivieren Die Vorderseiten werden nicht gezeichnet.

Schritt 10: Schattenvolumen zeichnen Die Rückseiten des Schattenvolumens werden gezeichnet und der Stencil Puffer wird an diesen Stellen um eins erniedrigt. Siehe Abbildung 20(c).

Schritt 11: Farbpuffer einschalten Um den Schatten zeichnen zu können, muss wieder in den Farbpuffer gemalt werden können. Außerdem wird der Tiefenpuffer nicht mehr gebraucht und sollte daher ausgeschaltet werden.

Schritt 12: Zeichne den Schatten Dazu benutzt man den Stencil Puffer einfach als Maske und zeichnet ein schwarzes bildschirmfüllendes Rechteck.

Mithilfe dieser zwölf Schritte, entsteht ein Schatten mithilfe des Shadow Volumes Algorithmus. Abbildung 21 zeigt den Bereich, der nach der Ausmaskierung mit dem Stencil Puffer übrig bleibt und den Schatten darstellt. Der Vorteil dieses Verfahrens ist, dass er Schattenkanten, ohne Artefakte, wie beispielsweise den Aliasing-Effekt beim Shadow Mapping erstellt. Allerdings muss ein Schattenvolumen für jedes Objekt in der Szene erstellt werden, so dass Geschwindigkeit des Algorithmus mit zunehmender Komplexität der Szene abnimmt.

4.5 Shadow Volumes in Augmented Reality-Umgebungen

Michael Haller präsentierte 2003 einen modifzierten Shadow Volumes Algorithmus, der für Augmented/Mixed Reality-Anwendungen genutzt werden kann [30]. Dieser Ansatz ist in drei Schritte gegliedert:

1. Reale Objekte werfen ihren Schatten auf reale Objekte
2. Virtuelle Objekte werfen ihren Schatten auf reale Objekte
3. Reale und virtuelle Objekte werfen ihren Schatten auf virtuelle Objekte

Schritt 1 Der erste Schritt besteht nur aus dem Kopieren des Kamerabildes in den Farbpuffer. Damit sind alle realen Objekte und gleichzeitig ihre Schatten im Farbpuffer.

Schritt 2 Im zweiten Schritt werden die Schatten der virtuellen auf die realen Objekte behandelt. Dazu werden zunächst die Phantomobjekte in den Tiefenpuffer geschrieben. Als nächstes werden die Shadow Volumes der virtuellen Objekte in den Stencil Puffer geschrieben. Danach wird der virtuelle Schatten auf die realen Objekte gemalt. Dazu werden die Schatten aus dem Stencil Puffer gerendert. Die Bereiche, in denen sich ein Schatten befindet, ersetzen damit die Bereiche des Kamerabildes.

Schritt 3 Um die Schatten der realen und virtuellen Objekte auf die virtuellen Objekte berechnen zu können wird zunächst die gesamte Szene (Phantom- und virtuelle Objekte) in den Farb- und in den Tiefenpuffer kopiert. Danach werden die Shadow Volumes für die virtuellen und die realen Objekte berechnet und in den Stencil Puffer kopiert. Dabei werden die Shadow Volumes aus dem zweiten Schritt überschrieben. Wenn im Wert des Stencil Puffers 0 steht, so wird die ganze Szene letztendlich gerendert.

Dieser Ansatz zeigt eine Umsetzung des Shadow Volumes Algorithmus für Mixed Reality-Umgebungen. Bei zu vielen virtuellen Objekten kann sich der Ansatz, aufgrund des zugrunde liegenden Shadow Volumes Algorithmus, verlangsamen, so dass die Echtzeitfähigkeit eventuell nicht mehr gewährleistet ist.

5 Konzeption

5.1 Einleitung

Um das Ziel, einen echtzeitfähigen Ansatz für Schatten in einer Mixed Reality-Umgebung zu entwickeln, erreichen zu können, sind verschiedene Vorüberlegungen notwendig. Die wichtigste Frage ist zunächst, welcher Schatten-Algorithmus sich mit wieviel Aufwand von einem reinen VR-Algorithmus für Mixed Reality-Umgebungen verändern lässt. Ist diese Frage geklärt, und man hat den passenden Algorithmus ausgewählt, soll zunächst versucht werden, ihn in Hinblick auf die Geschwindigkeit und auf das Aussehen zu verbessern. Dazu werden in den Kapiteln 5.4.1 und 5.4.2 einige Möglichkeiten vorgestellt. Danach folgt die Umsetzung des Algorithmus für die Mixed Reality. In Kapitel 5.4.3 werden daher wieder einige Konzepte für diese Umsetzung vorgestellt. Das Kapitel Konzeption stellt allerdings nur verschiedene Möglichkeiten vor, wie ein Schattenalgorithmus verbessert werden und für die Mixed Reality nutzbar gemacht werden kann. Erst das Kapitel Realisierung beschäftigt sich mit der Umsetzung des Ansatzes und beschreibt, welche Teile der Konzeption übernommen worden sind, und welche nicht.

Aus diesen Vorüberlegungen ist das Konzept der Arbeit entstanden. Dieses Konzept gliedert sich in folgende Teile und wird in den nächsten Unterkapiteln genauer vorgestellt:

- Es muss eine Testszene erschaffen werden, die es ermöglicht, die Vor- und Nachteile eines implementierten Schattenverfahrens schnell erkennen zu lassen.
- Der gewählte Standard Schattenalgorithmus muss beschleunigt werden, um die Echtzeitfähigkeit in einer AR-Umgebung zu gewährleisten.
- Der beschleunigte Algorithmus muss im Hinblick auf das Aussehen verbessert werden, insofern die Echtzeitfähigkeit das zulässt.
- Der verbesserte Schattenalgorithmus muss für eine Mixed Reality-Umgebung umgesetzt werden.
- Es soll ein anderes Verfahren implementiert werden, um den umgesetzten Ansatz im Hinblick auf die Echtzeitfähigkeit und evtl. dem Aussehen vergleichen zu können.

5.2 Vorüberlegungen

5.2.1 Echtzeitfähigkeit

Wie mit fast allen computergenerierten Bildern sind Schatten lediglich eine Approximation der realen Welt. Die Qualität dieser Approximation hängt von der Applikation ab, in der der Schatten generiert wird. In manchen Applikationen steht die Qualität des Schattens, z. B. ein realistischer weicher Schatten, im Vordergrund. Manchmal soll der Schatten aber auch so schnell wie möglich erzeugt werden, wobei die Qualität eher nebensächlich ist. Daher kann man sich zunächst fragen, welche Ergebnisse der perfekte Schattenalgorithmus in Bezug auf folgende Aspekte liefern würde [17]:

Geschwindigkeit Die Geschwindigkeit des Algorithmus sollte so schnell wie möglich sein. Eine virtuelle Simulation, z. B. einer Spielengine sollte mit mindestens 25 Frames pro Sekunde laufen, um einen ruckelfreien Ablauf zu gewährleisten. Das bedeutet für den Schattenalgorithmus, dass er für die Berechnung eines Bildes maximal 1/25 Sekunde Zeit hat. Allerdings gibt es mittlerweile so viele Effekte und aufwendige Algorithmen, dass nicht nur der Schattenalgorithmus in dieser Zeit berechnet werden muss. Daher sollte er so schnell wie möglich sein. Außerdem darf die Geschwindigkeit des Algorithmus nicht von Bewegung in der Szene beansprucht werden und alle Objekte, also Blocker, Empfänger und Lichtquellen sollten sich ohne Verzögerung frei in der Szene bewegen können.

Qualität Der Gesamteindruck der Schatten sollte gut sein. Artefakte sollten nicht vorhanden sein. Bei weichen Schatten sollte vor allem die Penumbra Region realistisch aussehen. Entscheidend ist, dass die Schatten nicht realistisch sein sollen, sondern nur realistisch aussehen.

Robustheit Die Ergebnisse des Algorithmus sollten richtig sein, unabhängig von der Platzierung der Szenenobjekte. Es sollten keine speziellen Einstellungen notwendig sein, um das richtige Ergebnis zu erzielen.

Allgemeingültigkeit Der Algorithmus sollte alle Arten von Geometrien der Objekte unterstützen. Er sollte z. B. nicht nur mit Dreiecken arbeiten können. Darüber hinaus sollte er auch die Selbstverschattung von Objekten unterstützen.

Im Zusammenhang mit den vier oben aufgeführten Aspekten kann man nun den Begriff der Echtzeitfähigkeit näher betrachten. Nach der Definition aus Kapitel 2.2 ist ein System echtzeitfähig, wenn es unter allen Bedingungen richtig und rechtzeitig auf alle auftretenden Ereignisse reagiert. Also sind die wichtigsten Aspekte eines echtzeitfähigen Schattenalgorithmus die Geschwindigkeit und die Robustheit. Allgemeingültigkeit und Qualität sind hierfür eher zu vernachlässigen, wobei es selbstverständlich wünschenswert ist, sie, solange es die Geschwindigkeit und die Robustheit zulassen, auch zu berücksichtigen.

Der Begriff der Robustheit, vor allem in AR-Umgebungen, erfordert einige Vorüberlegungen. Man benötigt verschiedene Vorkenntnisse über die Szene, so dass die Robustheit gewährleistet werden kann. Daher ist es wichtig, alle Möglichkeiten zu erfassen, wie Schatten in einer AR-Szene geworfen werden. Außerdem gibt es einige Probleme und daraus resultierende Einschränkungen, die es zu beachten gilt. Diese Vorüberlegungen werden in den nächsten beiden Unterkapiteln gemacht.

5.2.2 Wie können Schatten geworfen werden?

Eine der zentralen Fragen im Zusammenhang mit Schatten in Augmented Reality besteht darin, von welchem Typ (virtuell oder real) Lichtquelle, Blocker und Empfänger sein können, so dass Schatten geworfen werden. Nur mit Beachtung aller Fälle während der Implementierung, kann die Robustheit des Algorithmus gewährleistet werden und somit alle Möglichkeiten, in denen Schatten geworfen werden. Tabelle 1 beinhaltet diese Möglichkeiten. Fall 5 ist der Fall, der mit dem wenigsten Aufwand berücksichtigt werden muss. Alle Objekte sind real, d.h. sie werden von dem Kamerabild inklusive Schatten mitgeliefert, so dass man sie so übernehmen kann. Fall 2 wird in jedem oben beschriebenen Schatten-Algorithmus behandelt. Er stellt die typische VR-Szene dar, in der alle Objekte virtuell in der Szene vorhanden sind. Hier muss nur der ausgewählte Schattenalgorithmus angewandt werden. Die anderen sechs Fälle müssen bei der Implementierung eines Schatten-Algorithmus in einer AR-Anwendung auf jeden Fall genauer behandelt werden. Im Kapitel Realisierung werden diese sechs Fälle angesprochen.

Nr.	Lichtquelle	Blocker	Empfänger
1	virtuell	real	real
2	virtuell	virtuell	virtuell
3	virtuell	real	virtuell
4	virtuell	virtuell	real
5	real	real	real
6	real	virtuell	virtuell
7	real	real	virtuell
8	real	virtuell	real

Tabelle 1: Möglichkeiten, wie Schatten in AR-Umgebungen geworfen werden können

5.2.3 Probleme von Schatten in Augmented Reality

Um die im Kapitel 3 vorgestellten VR-Algorithmen auch für AR-Welten nutzen zu können, gibt es einige Probleme zu lösen und die daraus resultierenden Bedingungen zu berücksichtigen.

Probleme der Darstellungsmöglichkeiten Die beiden Darstellungsmöglichkeiten der Augmented Reality[2] haben auch im Bezug auf Schatten verschiedene Vor- und Nachteile. Beim optischen See-Through hat man nur die Möglichkeit der Überlagerung der Realität mit virtuellen Informationen. Man kann die Realität bzw. Teile davon nicht manipulieren oder ausschalten. Dies ist vor allem bei den Schatten problematisch. Da die Schatten normalerweise mit einer gewissen Transparenz arbeiten, so dass der Untergrund zum Teil noch zu sehen ist, werden beim optischen See-Through sehr viele Objekte sichtbar sein, die nicht mehr zu sehen sein dürften (siehe Kapitel 2.1.1). Überdeckt z.B. ein virtuelles Objekt ein reales Objekt, so wird durch die Transparenz des virtuellen Schattens der reale Schatten noch zu sehen sein. Auf diese Weise entstehen eine Vielzahl von „Geisterbildern“,

[2] Video- und optisches See-Through

die mit diesem Display nicht behandelt werden können. Im Jahr 2000 wurde ein Ansatz für ein anderes Display für die Augmented Reality vorgestellt, das dieses Problem lösen kann [24]. Dieses Display verbindet die Möglichkeiten des optischen mit denen des Video-See-Through auf zwei verschiedenen Ebenen. Dadurch wird die Realität durch eine Brille wahrgenommen, aber Teile davon können mit einer Maske verdeckt werden. Dadurch können die Geisterbilder verhindert werden und die virtuellen Information über der Verdeckung dargestellt werden. Allerdings hat sich dieses Display nicht durchgesetzt. Dieser Ansatz stellt eine sehr gute Möglichkeit dar, den Schattenalgorithmus auch mit optischen Displays nutzen zu können. Da dieses Display aber bei der Umsetzung dieser Diplomarbeit nicht zur Verfügung stand, wurde es nicht weiter behandelt. Aus diesem Grund ist der Ansatz dieser Diplomarbeit auch nur für Video-See-Through umgesetzt worden, weil eine realitätsnahe Darstellung von virtuellen Schatten, ohne Geisterbilder, mit einem normalen optischen Display nicht möglich ist. Video-See-Through hat das Problem der Geisterbilder nicht. Hier hat man die Möglichkeit, das von der Realität aufgenommene Kamerabild zu verändern. Im oben beschriebenen Fall kann man die Werte der Pixel, an denen der reale Schatten zu sehen ist, verändern, z. B. mit einer schwarzen Farbe überlagern, so dass das Geisterbild herausmaskiert wird.

Phantomobjekte Um einen Schatten von einem virtuellen auf ein reales Objekt werfen zu können, benötigt man die Geometrieinformationen dieses realen Objektes. Ohne diese Informationen würde der erstellte Schatten nicht korrekt auf dem Objekt erscheinen. Erst mithilfe einer virtuellen Nachbildung dieses realen Objektes, dem so genannten Phantomobjekt (engl: phantom object), kann der Schattenalgorithmus die Geometrie des realen Objektes berücksichtigen und einen korrekten Schatten erzeugen. Phantomobjekte sind einfach aufgebaute Meshes des realen Gegenstandes, die niemals auf den Bildschirm gemalt werden. Sie enthalten aber für den Schattenalgorithmus wichtige Informationen, wie die Eckpunkt-Positionen oder die Richtungen der Normalen. Ein Phantomobjekt ist mit einer 3D-Visualisierungs- und Animationssoftware wie z. B. Autodesk Maya[3] zu modellieren. Das modellierte Objekt speichert man am besten im VRML-Format ab, damit es beispielsweise das ARToolkit lesen kann.

Tracking Die Phantomobjekte helfen bei der Generierung der Schatten auf den realen Objekten. Allerdings müssen die Phantomobjekte direkt auf den realen Objekten platziert werden, so dass beide Geometrien übereinstimmen. Ansonsten passt der erstellte Schatten nicht mit den realen Objekten zusammen. Um dieses Problem zu lösen, gibt des das Tracking. Tracking bedeutet verfolgen, also die Position und Orientierung eines Gegenstandes oder einer Person zu erfassen.

Ein Beispiel für ein Trackinggerät ist ein GPS-Gerät. GPS-Satelliten senden ständig Signale aus, die von GPS-Empfängern erfasst werden um daraus ihre Position zu ermitteln. Je nach eingesetztem Tracking-Gerät ist die Genauigkeit unterschiedlich. GPS macht z. B. ein Grobtracking mit einer Genauigkeit von 100 Metern in 95 Prozent der gemessenen Fälle. Eine Feintracking Lösung bietet das optische Tracking. Hierzu werden Marker an der zu verfolgenden Person oder dem Gegenstand angebracht, die von Kameras verfolgt werden. Die Genauigkeit ist damit deutlich höher als beim Grobtracking, allerdings ist es mit viel Aufwand verbunden, ein Szenario mit Markern auszustatten.

Für das Tracking der virtuellen Objekte und der Phantomobjekte in einer realen Umgebung, bietet das Markertracking eine einfache und billige Lösung. Das ARToolkit [37]

[3] http://www.autodesk.com

verwendet einige vorgefertigte Marker, erkennt sie und gibt die Position des Markers innerhalb eines Videobildes zurück. Die Marker können dadurch mit virtuellen Objekten, also auch den Phantomobjekten, überlagert werden. Auf diese Weise können die Positionen der realen und der Phantomobjekt abgeglichen werden.

Reale Lichtquellen und reale Schatten Um einen Schattenalgorithmus ausführen zu können, benötigt man außerdem noch eine ungefähre Position der realen Lichtquellen. Auf diesen Positionen erstellt man virtuelle Lichtquellen, die ein weiterer essentieller Parameter für die Berechnung des virtuellen Schattens sind. Ohne diese Positionen muss man die virtuellen Lichter zufällig in der Szene anordnen und der virtuelle Schatten wird nicht in die gleiche Richtung wie der reale Schatten geworfen. Für dieses Problem gibt es noch keine zufrieden stellende Lösung. Es gibt einige bildverarbeitende Ansätze, die allerdings wiederum Rechenzeit benötigen und die Geschwindigkeit verringern. Daher wird in den meisten Ansätzen die Position der Lichtquellen als vorhanden vorausgesetzt.

Diese Probleme müssen erschweren die Arbeit mit der Augmented Reality. Allerdings gibt es für jedes der Probleme mehrere Lösungen. Der Ansatz dieser Diplomarbeit geht mit ihnen folgendermaßen um: Er arbeitet nur mit Video-See-Through Displays. Als Tracking benutzt er das Markertracking des ARToolkits. Die Phantomobjekte werden mit Maya modelliert, als VRML abgespeichert mit OpenVRML[4] in die Szene geladen. Die realen Lichtquellen werden nicht automatisch erkannt. Die virtuelle Lichtquelle wird in der Szene von Hand gesetzt, da bekannt ist, aus welcher Richtung das Licht kommt. Aus diesem Grund können in dem Ansatz auch nur vier der acht Fälle, wie Schatten in AR-Umgebungen geworfen werden können, berücksichtigt werden. Alle Fällen, in denen die Lichtquelle real ist, werden mit einer virtuellen Lichtquelle simuliert.

[4] http://openvrml.org/

5.3 Testszene

Eine Testszene, deren Hauptaufgabe es ist, einen Schattenalgorithmus im Hinblick auf Echtzeitfähigkeit und Aussehen zu testen, sollte im Groben folgende Kriterien erfüllen:

- Sie sollte mehrere Objekte beinhalten.
- Sie sollte Objekte beinhalten, deren Schattenkanten gut erkennbar sind.
- Die Objekte sollten sich selbst verschatten können.
- Sie sollte unterschiedliche Ecken und Kanten besitzen, um alle auftretenden Artefakte erkennen zu können.
- Sie sollte animiert werden, damit die Echtzeitfähigkeit getestet werden kann.
- Man sollte die Lichtquelle sehen und verschieben können, um die Schatten auch aus anderen Winkeln zu sehen.

Aus diesen Kriterien ist die Idee entstanden, Autos in einem teilweise geschlossenen Raum zu modellieren. Autos haben den Vorteil, dass sie eine sehr gut erkennbare Kontur besitzen. Außerdem haben ihre Karosserien eine Vielzahl an verschiedenen Ecken und Kanten. Des Weiteren bestehen Autos aus vielen verschiedenen Einzelteilen, so dass auch eine Selbstverschattung stattfinden wird. Der Raum besteht aus einem Boden und zwei Wänden, damit der Schatten auch auf den Seiten erkennbar ist. Die Szene soll insgesamt sehr einfach aufgebaut sein und nicht mit zu vielen Objekten überladen wirken, dass die unterschiedlichen Schatten nicht mehr einzeln zu erkennen sind.

5.4 Eigener Ansatz

Die Wahl des zugrunde liegenden Standard-Schattenalgorithmus für den Ansatz dieser Diplomarbeit ist auf das Shadow Mapping gefallen. Dies liegt hauptsächlich daran, dass eines der wesentlichsten Ziele des Ansatzes dieser Diplomarbeit die Echtzeitfähigkeit ist. Denn der größte Vorteil des Shadow Mappings im Gegensatz zu den Shadow Volumes ist, dass die Anzahl der Geometrien in der Szene unwichtig für die Geschwindigkeit des Algorithmus ist. Die Shadow Volumes müssen Schattenkanten für jedes einzelne Objekt in einer Szene berechnen, wodurch sich die Geschwindigkeit bei steigender Geometrieanzahl verringert.

Um den Standard Shadow Mapping-Algorithmus für eine Mixed Reality-Umgebung umsetzen zu können, werden zunächst einige Möglichkeiten vorgestellt, um die Geschwindigkeit und das Aussehen des Algorithmus zu verbessern. Danach erfolgen wiederum einige Ansätze für die Umsetzung in eine Mixed Reality-Umgebung.

5.4.1 Geschwindigkeit

Texturspeicherung Das Shadow Mapping hat einen entscheidenden Engpass, der die Geschwindigkeit des Algorithmus sehr stark einschränkt. Dieser Engpass liegt bei der Speicherung der Szene in einer Textur. Sowohl zu Anfang des Algorithmus, aber auch jedesmal, wenn sich die Lichtquelle bewegt, muss ein Tiefenbild der Szene aus Sicht der Lichtquelle erstellt werden. Dieses Tiefenbild wird dann in einer Textur, der Shadow Map gespeichert. Das Problem an diesem Vorgehen ist, dass die Szene dafür zunächst gerendert und anschließend noch in die Shadow Map kopiert werden muss. Der Vorgang, der dabei am meisten Zeit beansprucht, ist das Speichern in die Textur. Außerdem sind insgesamt zwei Schritte notwendig, um die Shadow Map zu füllen: das Rendern und das Speichern. Es wäre natürlich effizienter, wenn dies direkt in einem Schritt machbar wäre.

Um diesen Geschwindigkeitsverlust verringern zu können, gibt es zwei einfache Möglichkeiten. Die erste sind die *Framebuffer Objects*. Framebuffer Objects (FBO) stellen eine relativ neue Erweiterung in OpenGL dar. Sie haben ein einfaches Interface, um den Zielpuffer eines Renderprozesses zu ändern. Diese neu definierten Zielpuffer werden allgemein als *framebuffer-attachable images* bezeichnet und bieten die Möglichkeit, die Standardpuffer von OpenGL[5] zu ersetzen. Wird solch ein Bild an ein FBO gebunden, wird es von Fragmentoperationen ab sofort als Quelle und Ziel verwendet.

Der Hauptvorteil für das Shadow Mapping liegt aber schließlich in der Verwendung der *framebuffer-attachable images* als Texturen. Es ist mit Framebuffer Objects möglich, direkt in eine Textur zu rendern, ohne wie bisher üblich, zuerst in den aktiven OpenGL Framebuffer zu rendern und anschließend das Bild in die Textur zu kopieren. Dadurch ist der gesamte Vorgang, in eine Textur zu rendern, einfacher und effizienter.

Die zweite Möglichkeit ist ein Ansatz von Mark Harris [29]. Er entwickelte die Klasse *RenderTexture*. Diese Klasse unterstützt die Nutzung des *pbuffers*, der es, wie die Framebuffer Objects, ermöglicht, in andere Ziele als in das Fenster zu rendern. Für das Shadow Mapping sinnvoll sind die Ziele *Render To Texture* (RTT) oder *Render To Depth Texture* (RTDT), die wiederum das Rendern in eine Textur einfacher und effizienter gestalten.

[5] Farb-, Tiefen-, Accumulation- und Stencilpuffer

GPU Eine weitere Methode, um die Geschwindigkeit des Algorithmus zu verbessern, ist es, einzelne Teile des Algorithmus auf der Grafikkarte (GPU) ausführen zu lassen. Sinnvoll ist dies auf jeden Fall beim Tiefentest. Beim Tiefenwert muss der Tiefenwert aus der Shadow Map mit dem eigentlichen Tiefenwert aus der Szene verglichen werden. Wenn man die Shadow Map an den Fragment Shader übergibt, so kann dieser Tiefentest dort ausgeführt werden und das Ergebnis direkt auf die Farbe des Fragments Einfluss haben.

5.4.2 Aussehen

Aliasing Eines der größten Problem des Shadow Mapping-Algorithmus ist der Aliasing-Effekt. Dies sind treppenartige Abstufungen, die an den Rändern der Schatten entstehen. Er resultiert daraus, dass die Shadow Map, also die Textur, in der die Tiefenwerte aus Sicht der Lichtquelle gespeichert werden, eine bestimmte Auflösung hat. So kann der Tiefentest nur für jedes einzelne Pixel in der Shadow Map durchgeführt werden, was zu treppenartigen Abstufungen an den Schattenkanten führt.

Eine einfache Methode, den Aliasing-Effekt zu verringern ist es, die Auflösung der Shadow Map zu vergrößern. Wählt man beispielsweise anstelle von 512x512 Pixel eine Auflösung von 1024x1024, so wird der Tiefentest viermal so häufig durchgeführt. Dadurch wird der Aliasing-Effekt nicht eliminiert, aber die Schattenkanten erscheinen glatter. Allerdings muss sich bei der Erhöhung der Auflösung zeigen, wie viel Geschwindigkeit dabei verloren geht, und ob die Echtzeitfähigkeit darunter leidet.

Soft Shadow Um einen realistisch wirkenden Schatten darstellen zu können, dürfen weiche Schattenkanten nicht fehlen. Dadurch, dass in der Natur nur flächige Lichtquellen vorkommen (Kapitel 2.3), wirken die harten Schattenkanten des Shadow Mapping Algorithmus nicht sehr realistisch. Aus diesem Grund soll der Ansatz dieser Diplomarbeit einige einfache Umsetzungen für Soft Shadows berücksichtigen und wenn möglich mit einbeziehen.

Percentage Closer Filtering Das PCF-Shadow Mapping wurde bereits in Kapitel 4.2 beschrieben. Der Vorteil dieses Verfahrens ist, dass es nicht nur eine weiche Schattenkante erstellt, sondern dabei das Aliasing-Problem deutlich verringert. Dabei kann variiert werden, wie viele benachbarte Pixel der Algorithmus berücksichtigen soll. Je mehr Pixel benutzt werden, desto größer und unschärfer wird die Region der weichen Schattenkante. Allerdings muss auch bei diesem Verfahren die Geschwindigkeit berücksichtigt werden.

Percentage Closer Soft Shadow Einen weichen Schatten, der realistischer wirkt, erstellt das in Kapitel 4.3 vorgestellte Verfahren der Percentage Closer Soft Shadows. Hierbei wird der Abstand des Blockers zur Lichtquelle mit in die Berechnung eingerechnet, so dass weiche Schatten in Abhängigkeit zu diesem Abstand entstehen. Auch dieses Verfahren erstellt einen weichen Schatten und verringert das Aliasing-Problem, da es auf dem Percentage Closer Filtering aufbaut.

Allerdings ist dieses Verfahren auch das aufwendigste, der hier vorgestellten. Daher muss erst getestet werden, ob dieses Verfahren für einen echtzeitfähigen Ansatz in einer Mixed Reality-Umgebung verwendet werden kann.

5.4.3 Mixed Reality-Umgebung

Wenn einige Verbesserungen bezüglich der Geschwindigkeit und dem Aussehen des Algorithmus mit einbezogen worden sind, so ist der nächste Schritt eine Erweiterung des VR-Algorithmus für die Augmented Reality. Um das Shadow Mapping für AR-Umgebungen nutzbar zu machen, werden außer einigen virtuellen Objekten, Nachbildungen von realen Objekten benötigt. Das grundlegendste Problem dabei ist es, einen Schatten von virtuellen Objekten auf diesen Phantomobjekten anzuzeigen. Dies ist nicht so einfach, weil nur der Schatten auf den Phantomobjekten zu sehen sein darf, nicht aber das Objekt selbst. Für dieses Problem werden zwei verschiedene Ansätze vorgestellt:

Nutzung des Stencil Puffers Eine Lösung für das Problem ist die Nutzung des Stencil Puffers. Hier wird der Schatten zunächst in den Stencil Puffer gezeichnet, und erst am Ende über das gesamte Bild gemalt. Der Stencil Puffer bietet die Möglichkeit, nur die Teile des Puffers zu zeichnen, die beschrieben worden sind. Auf diese Weise sind die Phantomobjekte nicht sichtbar, aber der Schatten auf ihnen. Dieser Ansatz wurde bereits von Michael Haller für Shadow Volumes verwendet [30] und ist in Kapitel 4.5 vorgestellt worden.

Nutzung des Blendings Eine andere Methode für die Lösung des Problems, die bei der Konzeption dieser Diplomarbeit entstanden ist, benutzt die Blending-Funktion. Hierbei werden die Phantomobjekte vollständig gerendert. Allerdings werden Phantomobjekte, die im Licht liegen als transparent gezeichnet, so dass sie nicht zu sehen sind. Liegen sie im Schatten, so werden sie mit der Schattenfarbe gezeichnet. Das genaue Vorgehen ist dabei folgendes:

1. Kamerabild zeichnen
2. Shadow Map mit allen virtuellen Objekten erstellen
3. Szene nur mit virtuellen Objekten rendern
4. Blending aktivieren
5. Szene nur mit den Phantomobjekten rendern
6. Blending deaktivieren

Im Prinzip unterscheidet sich diese Methode nur in den Schritten 1 und 4-6 grundlegend vom Standard Shadow Mapping Algorithmus. Zunächst wird das Kamerabild für die Augmented Reality in den Framebuffer gezeichnet. Danach wird die Kamera an die Lichtposition transformiert und rendert die Szene. Entscheidend ist, dass nur die virtuellen, nicht die Phantomobjekte gerendert werden. Die Tiefenwerte der gerenderten Szene werden in der Shadow Map gespeichert. Danach wird die Kamera wieder an ihre ursprüngliche Position zurückgesetzt. Nun werden mit eingeschalteten Tiefentest alle virtuellen Objekte in der Szene gerendert. Durch diesen Renderschritt entsteht die Selbstverschattung der Objekte und die Schatten der virtuellen Objekte auf andere virtuelle Objekte. Um den Schatten der virtuellen Objekten auf den Phantomobjekten sichtbar zu machen, wird die Szene ein zweites Mal, allerdings nur mit den Phantomobjekten gerendert. Dafür muss allerdings zunächst das Blending aktiviert werden. Beim Rendern dürfen jetzt nur die Fragmente gezeichnet werden, die in den Schatten fallen. Daher wird der normale Tiefentest aus Formel 3 durchgeführt. Allerdings entscheidet das Ergebnis dieses Tiefentests nicht, ob das Fragment auf die Schattenfarbe gesetzt wird, oder die Farbe des Objektes erhält, sondern ob das Fragment auf die Schattenfarbe oder auf transparent gesetzt wird:

$$\begin{aligned} Z_B > Z_A &\Rightarrow \text{Fragment wird gezeichnet} \\ Z_B = Z_A &\Rightarrow \text{Fragment wird auf transparent gesetzt} \end{aligned} \tag{5}$$

Befindet sich das aktuelle Fragment im Schatten, so wird es wie gewohnt gezeichnet. Befindet es sich allerdings im Licht, so wird es auf transparent gesetzt, so dass dieser Teil des Objektes nicht sichtbar ist. Damit hat man genau den gewünschten Effekt, dass die virtuellen Objekte Schatten auf die Phantomobjekte werfen können, diese aber nicht zu sehen sind. Formel 5 zeigt diesen modifizierten Tiefentest.

5.4.4 Vergleichsalgorithmus

Um den Ansatz dieser Diplomarbeit mit einem anderen Schattenalgorithmus in einer Mixed Reality-Umgebung vergleichen zu können, soll der modifizierte Shadow Volumes-Algorithmus von Michael Haller [30] implementiert werden und zum Vergleich dienen (siehe Kapitel 4.5). Somit kann festgestellt werden, welches Verfahren sich besser für Mixed Reality-Umgebungen eignet. Entscheidend für diesen Vergleich sind wiederum die Aspekte der Echtzeitfähigkeit und des Aussehens des Schattens.

6 Realisierung

6.1 Einleitung

Da nicht alle Konzepte aus Kapitel 5 benutzt werden konnten, beschreibt das Kapitel Realisierung, welche davon übernommen worden sind und welche verworfen wurden. Dabei spielte wiederum der Aspekt der Echtzeitfähigkeit eine entscheidende Rolle. Des Weiteren geht das Kapitel auf einige wesentliche Teile der Implementierung ein und beschreibt deren Umsetzung. Zusätzlich befinden sich einige Auszüge aus dem entwickelten Quellcode für diesen Ansatz im Anhang A dieser Diplomarbeit.

6.2 Testszene

Um den Schattenalgorithmus in einer virtuellen Umgebung testen zu können, wurde in Maya eine kleine Autoszene modelliert. Sie besteht aus einem Untergrund, aus zwei Hauswänden, einem Verkehrsschild und zwei Spielzeugautos. Die Autos wurden animiert, so dass sie ständig im Kreis fahren. Eine Lichtquelle beleuchtet die gesamte Szene direkt von oben. Ein Schatten entsteht dadurch hauptsächlich von den Autos auf dem Boden. Bewegt man die Lichtquelle, so werden die Schatten der Autos und der des Schildes aber auch auf den Hauswänden zu sehen sein.

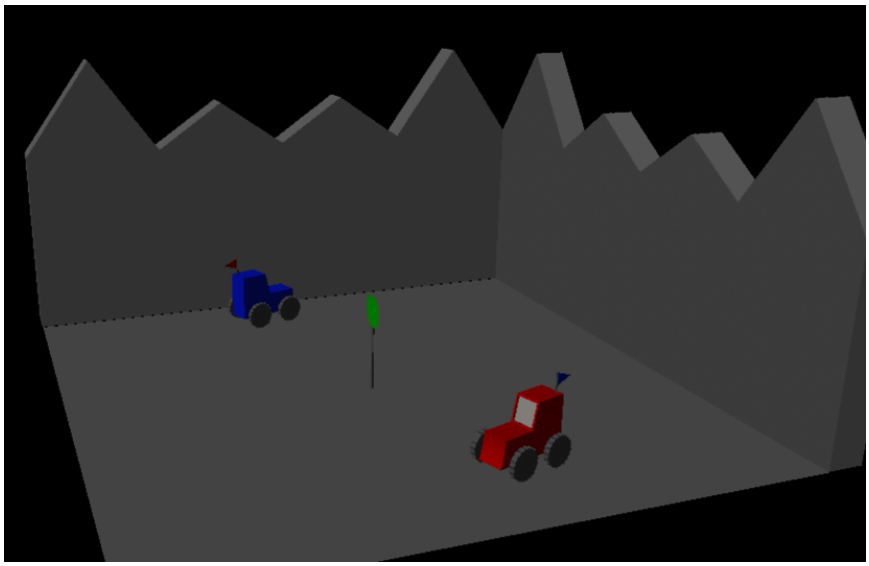


Abbildung 22: Testszene in Maya

6.3 Eigener Ansatz

6.3.1 Geschwindigkeit

Der Shadow Mapping Algorithmus wurde mit einigen Erweiterungen umgesetzt. Um das Tiefenbild aus der Lichtquelle nicht zuerst in den Framebuffer zu rendern und dann den Inhalt in eine Textur kopieren zu müssen, wurde die *Framebuffer Objects*-Erweiterung von OpenGL benutzt. Im Gegensatz zu dem vorgestellten Verfahren von Mark Harris hat sie den einfachen Vorteil, dass sie schon Bestandteil der OpenGL-API ist. Daher ist sie ohne viel Aufwand zu implementieren.

Um den Algorithmus weiter zu beschleunigen, wurde der Tiefentest auf der GPU ausgeführt. Hierfür wurde die OpenGL Shading Language (GLSL)[6] benutzt. Sie bietet die Funktion *shadow2DProj()*, die im Fragmentshader den Tiefentest ausführt, und somit das aktuelle Fragment direkt auf die Farbe des Objektes oder auf die Schattenfarbe setzen kann. Der Fragment- und der dazugehörige Vertexshader sind in Listing 1 und 2 in Anhang A dieser Arbeit zu finden.

Des Weiteren wurde das komplette Shading auf den Fragmentshader ausgelagert. Das beinhaltet die Berechnung der ambienten, diffusen und der spekularen Teile des Lichtes und des Emissionswertes, so dass diese nach der Schattenberechnung auf das Objekt angewendet werden konnten. Denn die Farbe eines Fragmentes, das im Schatten liegt, besteht nur aus dem ambienten Term des Lichtes und der Eigenemission, zusammen mit der Farbe des Schattens. Sie trifft kein diffuses und auch kein spekulares Licht.

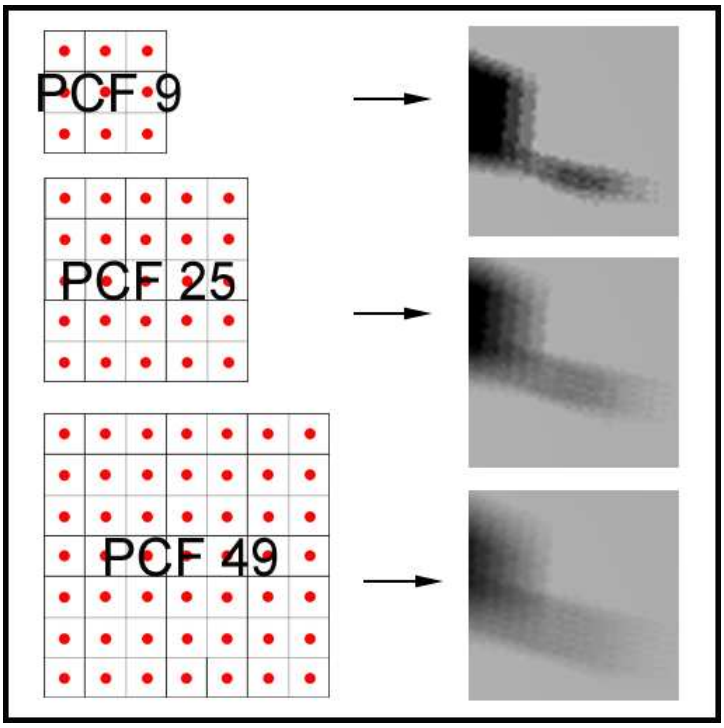


Abbildung 23: Verschiedene PCF-Filter

[6] http://www.opengl.org

6.3.2 Aussehen

Um eine geeignete Verbesserung zu finden, sind der PCF Filter und der PCSS-Ansatz, die in Kapitel 4.2 und 4.3 im Detail vorgestellt wurden, implementiert worden. Von dem PCF-Filter wurden drei Versionen umgesetzt, die den Tiefentest jeweils auf einer unterschiedlichen Anzahl von Nachbarpixeln ausführen. Die erste Version des Filters führt den Tiefentest auf 9 (PCF 9), die zweite auf 25 (PCF 25) und die dritte auf 49 (PCF 49) Pixeln aus. Diese Pixelanzahl ergibt sich aus den verschiedenen Radien, um das aktuelle Pixel im Fragmentshader. Ein Radius von eins von allen benachbarten Pixeln um das Ausgangspixel herum inklusive des Ausgangspixels ergeben eine Pixelanzahl von 9. Wird der Radius jeweils um einen Pixel nach außen erweitert kommt man bei einem Radius von zwei auf 25 und bei einem Radius von drei auf 49 Pixel. So können durch eine Schleife genau diese Anzahl an benachbarten Pixeln mit dem gewünschten Radius im Fragmentshader angesprochen werden. In Listing 3 im Anhang A dieser Arbeit ist der Fragment Shader des PCF-9-Filters abgebildet. Darin ist gut zu sehen, dass die Funktion *shadow2D*, die den Tiefentest durchführt, für das aktuelle und für die acht umliegenden Pixel, also insgesamt 9 Mal aufgerufen wird. Abbildung 6.3.1 zeigt die drei verschiedenen PCF-Filter und die Auswirkung auf den Schatten. Man sieht, dass bei steigender Anzahl der einbezogenen Pixel der weiche Bereich des Schattens immer größer wird und die Übergänge weiter verschwimmen.

Tabelle 2 zeigt die Geschwindigkeiten der einzelnen Verfahren in „Frames per Second“. Um die in Kapitel 2.2 beschriebene Echtzeitfähigkeit in diesem Ansatzes zu gewährleisten, musste ein Verfahren gewählt werden, dessen Frameraten deutlich über 25 fps liegen. Da die Frameraten beim Percentage Closer Soft Shadow (PCSS) Verfahren bei allen Auflösungen der Shadow Map immer unterhalb dieser Grenze bleiben, wurde dieser Algorithmus für den eigenen Ansatz verworfen. Allerdings führen auch die beiden PCF-Filter mit den größten Radien bei einer hohen Auflösung der Shadow Map zu keinen interaktiven Frameraten. Einzig der PCF Filter mit einem Radius von 9 Pixeln führt bei einer hohen Auflösung zu guten Frameraten. Da der Aliasing-Effekt nicht mehr zu sehen sein sollte, wurde deshalb der PCF-9-Filter mit einer Auflösung von 2048x2048 in den Ansatz mit einbezogen. Diese Lösung ist ein guter Kompromiss für einen weichen Schatten, mit kleinem Radius, ohne Artefakte, der nicht zu langsam ist.

	mit 512	mit 1024	mit 2048
Standard-SM	60 fps	59 fps	50 fps
PCF 9	59.9 fps	57.9 fps	45 fps
PCF 25	59.9 fps	29.9 fps	22 fps
PCF 49	30 fps	26 fps	14 fps
PCSS	20 fps	12 fps	8 fps

Tabelle 2: Geschwindigkeiten des Standard-Shadow Mapping Verfahrens, eines Percentage Closer Filtering mit verschiedenen Suchradien und des Percentage Closer Soft Shadow Verfahrens bei verschiedenen Auflösungen der Shadow Map

6.3.3 Mixed Reality-Umgebung

Um den beschleunigten PCF-Shadow Mapping-Algorithmus für eine Mixed Reality-Umgebung benutzen zu können, wurde der Ansatz mit dem Blending umgesetzt. Die Methode, die den Stencil Puffer nutzt, wurde verworfen. Letzteres Verfahren ist besser für Shadow Volumes geeignet, da dieser Algorithmus generell die Schatten in den Stencil Puffer schreibt. Für den Ansatz dieser Diplomarbeit wäre der Aufwand etwas größer gewesen, den Stencil Puffer als zusätzlichen Puffer zu benutzen. Aus diesem Grund wird das Verfahren mit dem Blending in dem Ansatz verwendet.

Abbildung 24 illustriert das Vorgehen des Ansatzes. In den Bildern rechts ist zu erkennen, was bei den einzelnen Funktionen gemacht bzw. zum finalen Bild hinzugefügt wird.

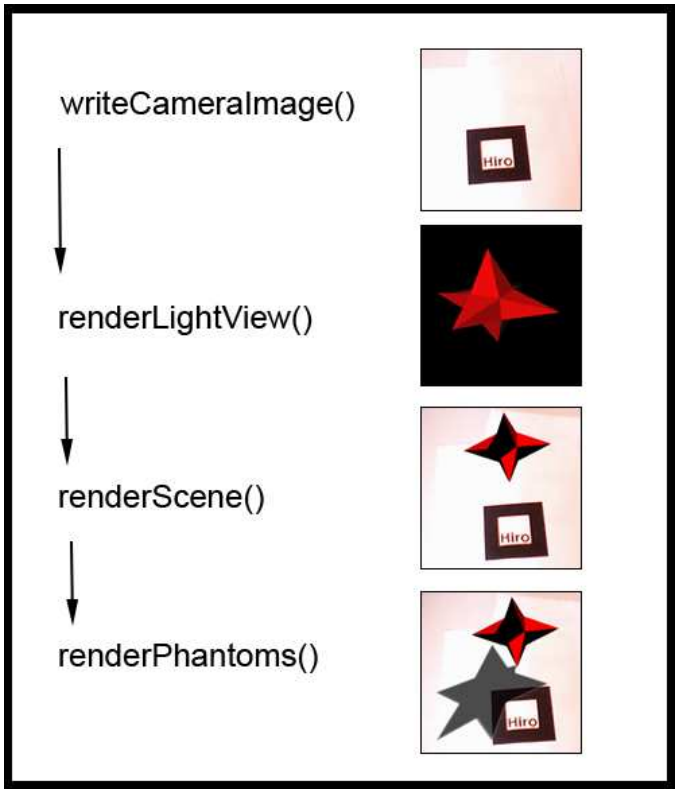


Abbildung 24: Vorgehensweise des eigenen Ansatzes: Das erste Bild von oben zeigt das aufgenommene Kamerabild. In Bild 2 ist das virtuelle Objekt aus Sicht der Lichtquelle zu sehen, wovon das Tiefenbild für die Shadow Map verwendet wird. Bilder 3 und 4 fügen den virtuellen Schatten und das virtuelle Objekt in das Kamerabild ein. Dafür wird zuerst das Objekt gemalt, dann die Selbstverschattung und dann die Schatten auf andere Objekte.

Die Funktion *writeCameraImage()* schreibt das Kamerabild in den Framepuffer. Das AR-Toolkit ist dabei für das Erkennen der Marker im Kamerabild zuständig. Als nächstes erstellt *renderLightView()* die Shadow Map aus Sicht der Lichtquelle. Allerdings sind dabei nur die virtuellen Objekte zu sehen, nicht aber die Phantomobjekte. Das zweite Bild in Abbildung 24 zeigt ein gerendertes Bild aus Sicht der Lichtquelle. Hiervon werden nur die Tiefenwerte in die Shadow Map geschrieben. Für das eigentliche Rendern der virtuellen Objekte werden zwei verschiedene Methoden verwendet: *renderScene()* und *renderPhantoms()*. Die erste Methode beinhaltet alle virtuellen Objekte der Szene und die zweite nur die Phantomobjekte. So können beim Aufrufen der Methoden verschiedene Shader verwendet werden. Die Funktion *renderScene()* rendert somit alle virtuellen Objekte inklusive aller Schatten von anderen virtuellen Objekten und sich selbst. Die Funktion *renderPhantoms()* rendert den Schatten der virtuellen Objekte. Dazu wird der Fragment Shader aus Listing 4 benutzt. Hier wird berechnet, ob sich das aktuelle Fragment im Schatten, oder im Licht befindet. Befindet es sich im Schatten, so bekommt es die aktuelle Schattenfarbe, ist es allerdings im Licht, so wird seine Transparenz auf 1 gesetzt, so dass es nicht sichtbar ist. Abbildung 25 verdeutlicht diesen Ablauf. Ist das Fragment im Schatten, so wird die Schattenfarbe auf dem Phantomobjekt gezeichnet, so dass im Endeffekt der Schatten des Sterns zu sehen ist. Im umgekehrten Fall, wird nur der Teil des Bildes ohne den virtuellen Schatten gemalt, allerdings mit der Transparenz auf 1, so dass der Rest des Bildes wieder zu sehen ist. Die Teile im Bild, die jeweils gezeichnet werden, sind in Abbildung 25 mit gelb gekennzeichnet.

Wichtig ist, dass dieser Fragment Shader nur mit vorher eingeschaltetem Blending funktioniert, da sonst die Transparenzen nicht erkannt werden. Außerdem sollte vor der Ausführung das Backface-Culling aktiviert werden. Ist es deaktiviert, so werden die Schatten der virtuellen Objekte sowohl auf den Vorderseiten, als auch den Rückseiten der Phantomobjekte zu sehen sein, was zwar einen interessanten, aber nicht den gewünschten Effekt erzielt.

Mit diesem Ansatz sind allerdings nur 4 der 8 Möglichkeiten, wie Schatten in AR Umgebungen geworfen werden können, umgesetzt worden. Für alle Fälle, in denen die Lichtquelle real ist, muss zuerst die Lichtquelle bzw. die Richtung des Lichtes aus dem Bild detektiert werden. Dazu sind verschiedene Methoden der Bildverarbeitung notwendig, die in diesem Ansatz nicht eingebaut sind. Der Ausblick in Kapitel 8 geht etwas genauer auf diese Möglichkeiten ein.

6.3.4 Vergleichsalgorithmus

Der Shadow Volume Algorithmus wurde wie in Kapitel 4.5 beschrieben umgesetzt. Hierfür sind wiederum zwei Renderfunktionen benötigt worden. Eine beinhaltet nur die virtuellen Objekte und eine nur die Phantomobjekte. Der Unterschied zu dem eigenen Ansatz besteht neben der Tatsache, dass der Algorithmus Shadow Volumes als zugrunde liegenden Schattenalgorithmus verwendet, in der Benutzung des Stencil Puffers. Er bietet eine andere Möglichkeit die Schatten der virtuellen Objekte, die auf die realen Objekte fallen sollen, zu berechnen und letztendlich über das Bild zu legen.

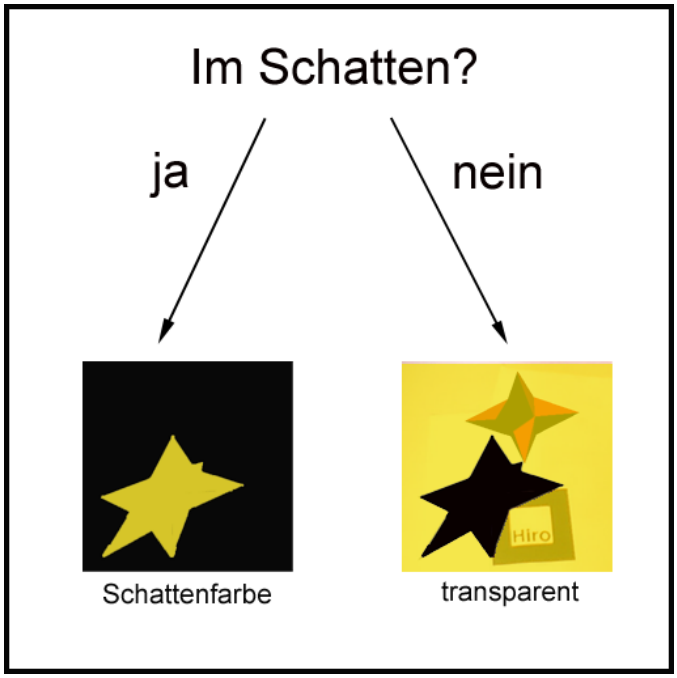


Abbildung 25: Vorgehensweise des Fragmentshaders während der Funktion *renderPhantoms()*: Die Bereiche in den Bildern werden jeweils gemalt. Wenn das Fragment im Schatten liegt, wird die Schattenfarbe verwendet, andernfalls wird das Fragment auf transparent gesetzt.

6.4 Probleme mit den Koordinatensystemen

Ein generelles Problem beim Shadow Mapping besteht darin, dass man zwischen der Lichtquelle und der Kamera transformieren muss. In einer normalen VR-Umgebung entstehen dabei keine Probleme, da beide Positionen im gleichen Koordinatensystem vorliegen. Eine Transformation der Kamera an die Position der Lichtquelle kann beispielsweise erfolgen, indem zunächst die Lichtquelle in den Ursprung verschoben wird und danach mit den Koordinaten der Lichtquelle an ihre Position.

In einer Mixed Reality-Umgebung ist dies etwas komplizierter. Das liegt daran, dass das ARToolkit und OpenGL nicht das gleiche Koordinatensystem benutzen. Hier muss man zwischen verschiedenen Koordinatensystemen unterscheiden. Das erste ist das Kamerakoordinatensystem (KCS). Wenn ein Marker im Videobild erkannt wird, liefert er dem ARToolkit seine Position in Kamerakoordinaten zurück. Es ist also das Koordinatensystem der echten Kamera, die das Videobild aufgenommen hat. Der getrackte Marker hat auch ein Koordinatensystem, damit Objekte relativ zu ihm beschrieben werden können. Dies ist das Markerkoordinatensystem (MCS). Wird in OpenGL ein Objekt positioniert, so liegt dies im OpenGL Koordinatensystem. Dazu sagt man aber auch Weltkoordinatensystem (WCS).

Das bei dem Ansatz aufgetretene Problem ist, dass die Kamera und die Lichtquelle in unterschiedlichen Koordinatensystemen vorliegen. Die Lichtquelle wird in OpenGL einfach festgelegt und liegt daher in Weltkoordinaten vor. Da das ARToolkit das Koor-

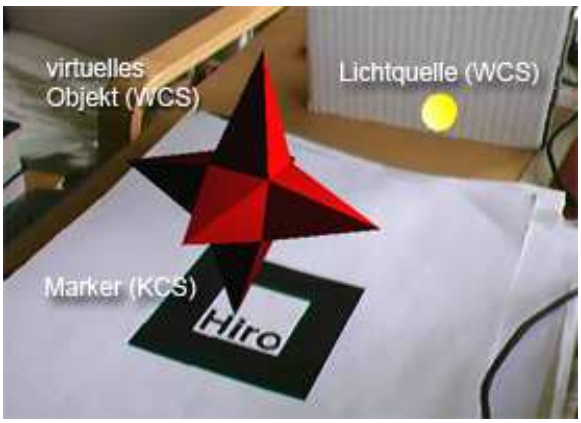


Abbildung 26: Unterschiedliche Koordinatensysteme

dinatensystem des Markers mit der gleichen Orientierung versieht, wie das OpenGL Koordinatensystem, ist dies gleichzeitig das Markerkoordinatensystem. Allerdings wird die Kameraposition durch das Tracking des Markers festgelegt. Diese liegt in Kamerakoordinaten vor. Um die Kamera aber für das Shadow Mapping an die Position der Lichtquelle setzten zu können, muss die Position der Kamera von Kamera- in Weltkoordinaten überführt werden, und danach wieder zurück.

Die Lösung dieses Problems liefert das ARToolkit selbst. Sie gibt nicht nur die Position der Kamera in Kamerakoordinaten zurück, sondern mit einer anderen Funktion auch ihre Modelview- und Projektionsmatrizen. Speichert man diese und übergibt sie an die Schattenklasse, so kann die Transformation in ein anderes Koordinatensystem übergangen werden. Beim Shadow Mapping kann also für die Kamera zunächst die Position der Lichtquelle mit den entsprechenden Matritzen geladen werden. Eine Rücktransformation ist dann mit den Matritzen der Kamera möglich. Auf diese Weise umgeht man eine Transformation der Position der Kamera zunächst in Weltkoordinaten und wieder zurück in Kamerakoordinaten.

7 Ergebnisse

Das Kapitel Ergebnisse dokumentiert den Algorithmus im Hinblick auf die Geschwindigkeit und das Aussehen. Das letzte Unterkapitel stellt den umgesetzten Ansatz mit dem Shadow Mapping dem Shadow Volumes-Ansatz gegenüber. Getestet wurde auf einem Pentium 4, 3.2 GHz, 1 GB RAM mit NVIDIA GeForce 6600 GT Grafikkarte.

7.1 Geschwindigkeit

Um die Geschwindigkeit vergleichen zu können, wurde sie nach jeder eingebauten Verbesserung gemessen. Zunächst wurde die Framerate beim Standard Shadow Mapping-Algorithmus gemessen. Dieser wurde, wie in Kapitel 6 beschrieben, zunächst mit Framebuffer Objects realisiert. Danach wurde der Tiefentest auf die GPU ausgelagert und zum Schluss wurde der Algorithmus um das Percentage Closer Filtering erweitert. Tabelle 3 zeigt die Frameraten der einzelnen Schritte des Algorithmus, die auf diesem Computer erzielt wurden. Dabei ist zu erwarten, dass die Verbesserung mit den Framebuffer Objects eine deutlichere Geschwindigkeitssteigerung hervorruft, als die Umsetzung des Tiefentests auf der GPU. Der Algorithmus des PCF-Shadow Mapping sollte das Ergebnis dabei etwas verlangsamen.

Genau wie erwartet, steigt die Zahl der Frames per Second bei der Beschleunigung des Standard Shadow Mapping-Algorithmus mit den Framebuffer Objects von 70 auf 85 fps. Auch die Verlagerung des Tiefentests auf die GPU beschleunigt den Algorithmus. Die Percentage Closer Filtering-Erweiterung verlangsamt den Ansatz etwas, allerdings werden nach wie vor interaktive Frameraten erreicht. Dies war auch der Grund, warum das Percentage Closer Filtering und nicht das Percentage Closer Soft Shadow-Verfahren verwendet, denn mit dem letzteren Verfahren sind die Frameraten bis teilweise unter 10 fps gesunken.

Standard-SM	mit FBO	mit GPU	mit PCF
70 fps	85 fps	90 fps	60 fps

Tabelle 3: Geschwindigkeiten des Ansatzes mit den jeweiligen Verbesserungen

7.2 Aussehen

7.2.1 Virtuelle Umgebung

Um das Aussehen des Ansatzes bewerten zu können, wurden zunächst verschiedene Bilder der virtuellen Szene gemacht. Dabei wurde die Auflösung der Shadow Map verändert und das Percentage Closer Filtering angewendet. Auf diese Weise kann man die Vor- und Nachteile des eigenen Ansatzes sehen. Abbildung 27 zeigt die Szene mit dem Standard Shadow Mapping-Verfahren. Links im Bild sieht man die gesamte Szene. Hier ist die Auflösung benutzt worden, die auch im eigenen Ansatz verwendet wird (2048x2048). Die drei Bilder rechts der Szene zeigen einen Ausschnitt, mit jeweils unterschiedlicher Auflösung der Shadow Map. Ganz oben der Ausschnitt hat eine Auflösung von 512x512, in der Mitte von 1024x1024 und unten von 2048x2048. Hier ist gut zu erkennen, dass der Aliasing-Effekt mit höherer Auflösung der Shadow Map deutlich geringer wird. Außerdem können bei geringerer Auflösung Fehler im Schatten auftreten, wie oben rechts im Bild zu sehen ist.

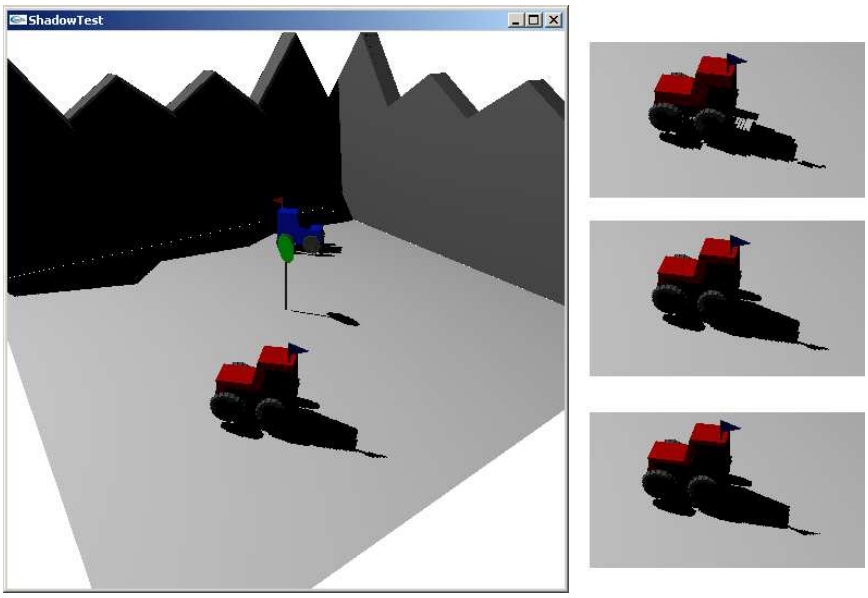


Abbildung 27: Links sieht man die gesamte Szene, in der Auflösung, die im eigenen Ansatz verwendet wird. Die anderen drei Bilder zeigen einen Ausschnitt der Szene, der mit unterschiedlicher Auflösung der Shadow Map gerendert wurde. Oben: 512x512. Mitte: 1024x1024. Unten: 2048x2048.

Abbildung 28 zeigt die Szene mit der Percentage Closer Filtering-Erweiterung. Für alle Bilder wurde die ausgewählte Shadow Map-Auflösung von 2048x2048 genommen. Links im Bild ist wiederum die gesamte Szene zu sehen. Hier wurde der PCF-Filter des eigenen Ansatzes verwendet, der zusätzlich für die 8 benachbarten Pixel den Tiefentest durchführt. Die drei Bilder rechts zeigen den PCF-Filter mit unterschiedlichem Radien. Oben wurde auf 9, in der Mitte auf 25 und unten auf 49 Pixeln der Tiefentest durchgeführt. Je größer dieser Suchradius gewählt wird, desto größer wird der unscharfe Randbereich und desto weicher wird der Schatten. Allerdings ist der Ansatz bei Benutzung des PCF-49-Filters im Vergleich zum PCF-9-Filter auch deutlich langsamer (siehe Tabelle 2). Daher ist auf einen sehr weichen Schatten verzichtet worden.

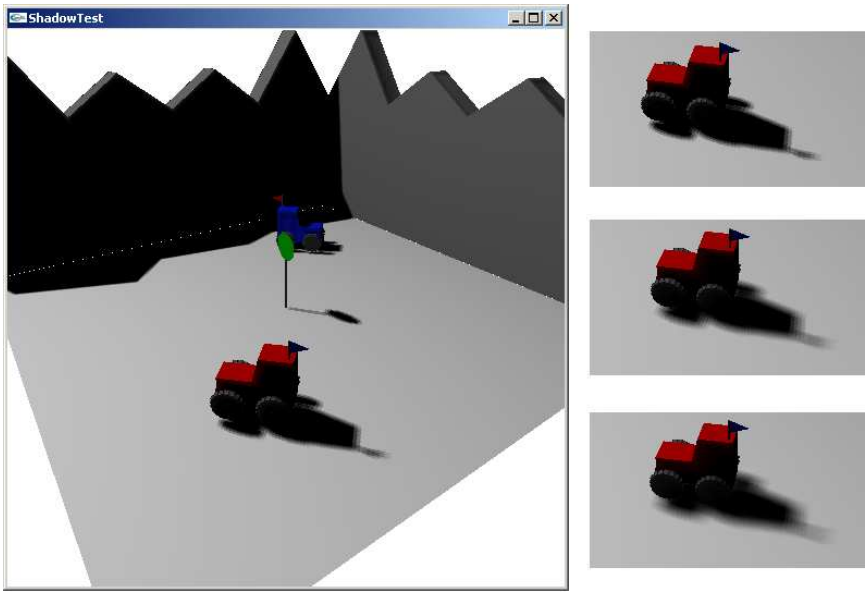
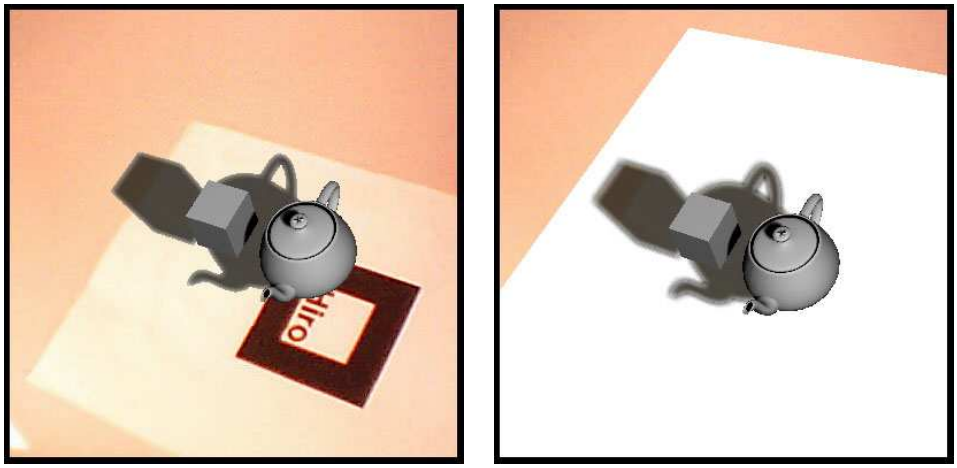


Abbildung 28: Links sieht man die gesamte Szene, mit dem PCF-Filter, der im eigenen Ansatz verwendet wird. Die anderen drei Bilder zeigen einen Ausschnitt der Szene mit unterschiedlich großen Suchradien des PCF-Filters. Oben wird zusätzlich auf den 8 benachbarten Pixeln der Tiefentest durchgeführt, in der Mitte auf 16 und unten auf 32.

7.2.2 AR-Umgebung

Die folgenden Bilder zeigen den umgesetzten Ansatz dieser Diplomarbeit in einer einfach aufgebauten AR-Umgebung. Abbildung 29 zeigt eine reale Umgebung, in die eine virtuelle Teekanne und ein virtueller Würfel hinzugefügt werden. In Abbildung 29(a) entsteht der Eindruck, dass die virtuellen Objekte ihren Schatten auf den realen Tisch werfen. In Abbildung 29(b) ist zusätzlich ein Phantomobjekt eingeblendet. Hierbei handelt es sich um eine virtuelle Fläche, die auf den Tisch gelegt wurde. Der Schatten der virtuellen Objekte wird also nicht direkt auf dem realen Tisch projiziert, sondern nur auf die virtuelle Fläche. Man sieht aber nicht nur den Schatten der virtuellen Objekte auf dem Tisch. Auf dem Deckel der Teekanne ist die Selbstverschattung zu erkennen. Außerdem wirft die Teekanne einen kleinen Schatten auf die untere Ecke des Würfels.



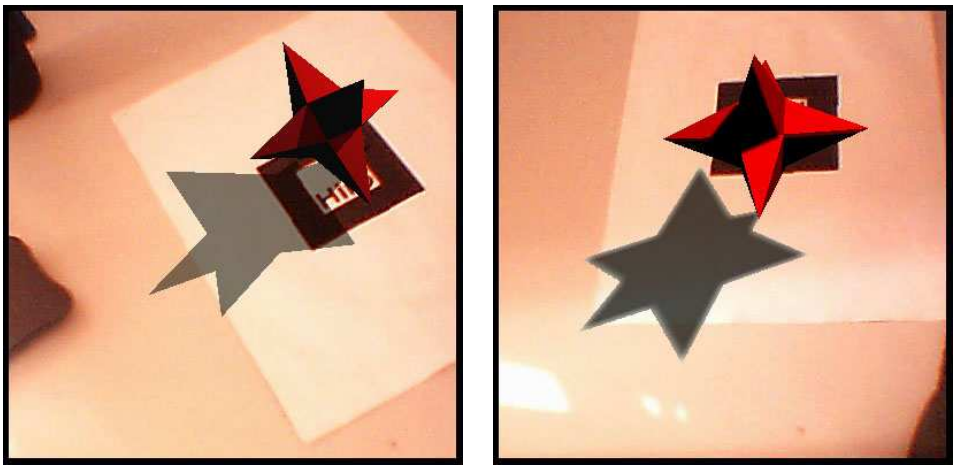
(a) Der Schatten der virtuellen Objekte fällt auf den realen Tisch

(b) Der Schatten der virtuellen Objekte fällt auf ein Phantomobjekt

Abbildung 29: Schatten in Augmented Reality

7.3 Vergleich

In diesem Unterkapitel soll der eigene Ansatz mit dem Shadow Volumes-Ansatz von Michael Haller [30], der in Kapitel 4.5 vorgestellt wurde, verglichen werden. Es wird wiederum sowohl auf die Geschwindigkeit, als auch auf das Aussehen geachtet. Abbildung 30 zeigt eine Szene, die links mit dem Shadow Volumes-Ansatz umgesetzt wurde und rechts mit dem Shadow Mapping-Ansatz dieser Diplomarbeit.



(a) Ansatz der Shadow Volumes (b) eigener Ansatz mit Shadow Mapping

Abbildung 30: Vergleich von Shadow Volumes in AR mit dem eigenen Ansatz

7.3.1 Geschwindigkeit

Die Geschwindigkeit beider Ansätze wurde zunächst in der Szene, die in Abbildung 30 zu sehen ist, gemessen. Beim Shadow Volumes-Ansatz wurden 25 fps und beim eigenen Ansatz 23 fps gemessen. In einer sehr einfachen Szene ist die Geschwindigkeit daher ungefähr gleich. Daher wurde die Szene mit 20 weiteren virtuellen Objekten erweitert. Das Ergebnis ist eindeutig. Während der eigene Ansatz mit 20 fps nicht deutlich an Geschwindigkeit verliert, läuft der Shadow Volumes Ansatz nur noch mit 7 fps. Das ist das grundlegende Problem des Shadow Volume-Algorithmus. Denn es muss für jedes Objekt ein Schattenvolumen berechnet werden. Daher wird er bei komplexeren Szenen immer langsamer.

Allerdings hat der Shadow Volumes-Ansatz den Vorteil, dass er auf gewöhnlichen Grafikkarten läuft. Er benötigt lediglich OpenGL. Der eigene Ansatz läuft nur auf einer Grafikkarte mit programmierbarem Vertex- und Fragmentshader. Daher gibt es teilweise Probleme auf handelsüblichen Notebooks.

Die Frameraten unterscheiden sich so deutlich von denen aus Tabelle 3, da bei diesen Messungen die verschiedenen Algorithmen der AR-Umgebung, wie das Tracking und das Anzeigen des Kamerabildes, zusätzlich zu dem Schattenalgorithmus benutzt wurden.

7.3.2 Aussehen

Tabelle 4 stellt die wichtigsten Aspekte beider Verfahren gegenüber. Zunächst ist festzustellen, dass beide Verfahren jeweils korrekte Schatten mit Selbstverschattung werfen. Außerdem sind bei beiden Verfahren keine groben Artefakte an den Schattenkanten zu erkennen.

Der einzige erkennbare Unterschied ist, dass der eigene Ansatz weiche Schattenkanten erzeugt. Diese sind nicht so schön wie in der Realität, aber sie lassen den eigenen Ansatz dadurch etwas realistischer wirken, als den Ansatz mit den Shadow Volumes.

	Selbstverschattung	Artefakte	weicher Schatten
Shadow Volumes	ja	nein	nein
eigener Ansatz	ja	nein	ja

Tabelle 4: Vergleich der Schattenarten beider Ansätze

7.3.3 Fazit

Der Vergleich des eigenen Ansatzes mit dem Shadow Volumes-Ansatz von Michael Haller hat gezeigt, dass beide Verfahren generell gut für den Einsatz in Mixed Reality-Umgebungen geeignet sind. Allerdings wird es bei zunehmender Anzahl an Polygonen mit dem Ansatz der Shadow Volumes schwierig, interaktive Frameraten zu bekommen. Der eigene Ansatz hat diese Beschränkungen aufgrund des zugrunde liegenden Shadow Mapping-Algorithmus nicht. Die Frameraten sinken bei zunehmender Komplexität der Szene nur gering. Außerdem erzeugt er aufgrund des PCF-Filters weiche Schattenkanten, die den Realitätsgrad des Schattens deutlich erhöhen. Des Weiteren hat er den Vorteil, dass die Frameraten schnell auf Kosten des Aussehens erhöht werden können, indem man den PCF-Filter nicht benutzt. Somit ist der vorgestellte Ansatz dem Shadow Volumes-Ansatz sowohl in der Geschwindigkeit als auch im Aussehen überlegen.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In der Diplomarbeit *Echtzeitfähige Schatten in Mixed Reality-Umgebungen* wurde ein Ansatz vorgestellt, der den Standard Shadow Mapping-Algorithmus so modifiziert, dass er auch für Mixed Reality-Umgebungen genutzt werden kann. Die Ergebnisse des Ansatzes sind überzeugend. Der präsentierte Algorithmus erweitert Augmented Reality-Anwendungen um einen realistisch aussehenden Schatten, ohne dabei sehr langsam zu werden. Dadurch wirken alle virtuellen Objekte in der realen Szene deutlich realistischer. Außerdem ist es für die Benutzer leichter, die Distanzen und die Größen der virtuellen Objekte einzuschätzen, wodurch sie einfacher mit ihnen interagieren bzw. sie manipulieren können.

Der eigene Ansatz besteht darin, dass er den Shadow Mapping-Algorithmus von Lance Williams [25] weiterentwickelt. Der Standard Shadow Mapping-Algorithmus benutzt zwei Renderschritte, um einen virtuellen Schatten zu erstellen. Im ersten Schritt wird die Szene aus Sicht der Lichtquelle gerendert. Die Tiefenwerte des gerenderten Bildes werden dann in einer Textur, der Shadow Map gespeichert. Im zweiten Schritt wird die Szene erneut, allerdings aus Sicht der Kamera gerendert. Hierbei wird die Shadow Map als projektive Textur auf die Szene projiziert. So kann ein Tiefentest durchgeführt werden, der den Tiefenwert aus der Shadow Map mit dem Tiefenwert aus Sicht der Kamera vergleicht. Sind die beiden Tiefenwerte gleich groß, so liegt das Pixel im Licht. Ist der Wert aus der Shadow Map kleiner, so liegt das Pixel im Schatten und bekommt eine dunklere Farbe.

Um den Standard Shadow Mapping Algorithmus zu beschleunigen wurde die OpenGL Erweiterung *Framebuffer Objects* verwendet. Um das Tiefenbild in der Shadow Map zu speichern, muß normalerweise zunächst das Bild gerendert werden um dann den Inhalt in eine Textur kopieren zu können. Dieser Schritt kann somit umgangen werden, da die Framebuffer Objects die Möglichkeit bieten direkt in eine Textur zu rendern. Um den Algorithmus weiter zu beschleunigen wurde außerdem der Tiefentest auf den Fragment Shader ausgelagert.

Eines der Hauptprobleme des Shadow Mappings, das Aliasing, wurde mit dem Ansatz des Percentage Closer Filtering [58] gelöst. Dieser treppenartige Effekt an den Schattenkanten wird weniger, da ein PCF-Filter den Schattentest nicht nur auf einem, sondern zusätzlich auf allen benachbarten Pixeln ausführt. Das Ergebnis wird durch die Anzahl der durchgeführten Schattentests dividiert. Auf diese Weise entstehen Pixel mit einer Farbe, die zwischen der Schattenfarbe und der Farbe des Objektes liegt. Das Resultat sind weiche, etwas verschwommene Schattenkanten, so dass der Aliasing-Effekt verringert wird.

Der Ansatz arbeitet mit einem PCF-9-Filter und einer Auflösung von 2048x2048 Pixel. Dieser modifizierte Shadow Mapping-Algorithmus ist weiter verändert worden, damit er für die Augmented Reality genutzen werden kann. Als Trackingsystem dient dabei das ARToolkit. Um den Schatten von virtuellen auf realen Objekten allerdings sehen zu können, benötigt man Phantomobjekte, die auf die realen Objekte gelegt werden, damit der Schattenalgorithmus eine Geometrie hat, auf der er den Schatten erstellen kann. Die Vorgehensweise des Algorithmus ist der folgende: Zuerst wird das Videobild in den Farbpuffer kopiert. Das ARToolkit ist dabei für das Erkennen der Marker im Kamerabild zuständig. Als zweites wird die Shadow Map aus Sicht der Lichtquelle erzeugt. Allerdings sind hierbei nur die virtuellen Objekte sichtbar. Im dritten Schritt werden alle virtuellen Objekte gerendert. Der Tiefentest wird ausgeführt, so dass die Selbstverschattung der Objekte entsteht und die Schatten von den virtuellen auf andere virtuelle Objekte. Im letzten Schritt werden nur die Phantomobjekte gerendert. Hierbei wird ein anderer Fragment Shader ver-

wendet. Dieser führt zunächst den normalen Schattentest erneut aus. Liegt das Fragment im Schatten, so bekommt es die Farbe des Schattens. Liegt es allerdings im Licht, so wird das Fragment auf transparent gesetzt, so dass es nicht gesehen wird. Auf diese Weise wird nur der Schatten der virtuellen Objekte auf den Phantomobjekten und somit auf den realen Objekten gezeichnet. Die nicht verschatteten Teile der Phantomobjekte sind nicht zu sehen.

Der Ansatz liefert einen korrekten weichen Schatten in Augmented Reality und läuft mit 50 fps auf einem Pentium 4 mit 3.2 GHz, 1 GB RAM und einer NVIDIA GeForce 6600 GT Grafikkarte. Der große Vorteil dieses Verfahrens ist, dass mit zunehmender Komplexität der virtuellen Szene die Geschwindigkeit des Schattenalgorithmus nicht abnimmt. Durch die Benutzung der Percentage Closer Filtering-Erweiterung, besteht die Möglichkeit auf Kosten der Geschwindigkeit den Schatten noch weicher zu machen, indem auf mehr Randpixel der Schattentest durchgeführt wird. Um Geschwindigkeit zu gewinnen, so kann der PCF-Filter auch weggelassen werden, was den Realitätsgrad des Schattens allerdings wieder sinken läßt. Ein Vergleich mit einem für die Augmented Reality nutzbaren Shadow Volumes Algorithmus in komplexen Szenen zeigt, dass der eigene Ansatz einen weicheren und schöneren Schatten in kürzerer Zeit wirft.

8.2 Ausblick

8.2.1 Verbesserungen

Der Ansatz hat gezeigt, dass er in geringer Zeit einen korrekten Schatten in einer Mixed Reality-Umgebung werfen kann. Allerdings gibt es immer noch viele Möglichkeiten diesen Ansatz zu verbessern und zu erweitern.

Verbesserung des Shadow Mappings Es gibt viele Paper, die sich mit der Verbesserung des Shadow Mapping Algorithmus beschäftigen. Dabei geht es meistens um die Beseitigung von Artefakten oder um die Erschaffung eines weichen Schattens. Einige dieser Ansätze wurden in Kapitel 3.3.1 vorgestellt. Allerdings existieren deutlich mehr Lösungen für diese Probleme. Daher wäre es möglich, den Schatten des Ansatzes durch eine andere Herangehensweise als den PCF-9-Filter noch realistischer erscheinen zu lassen. Eine weitere Verbesserung wäre es auch, mehrere bzw. andere Lichtquellen im Algorithmus zu unterstützen. Der Shadow Mapping-Algorithmus unterstützt nur Spotlichtquellen, da eine einfache Shadow Map nicht die gesamte Hemisphere um sich herum aufnehmen kann. Daher wäre eine Integration von Punkt- oder von flächigen Lichtquellen wünschenswert.

Erkennung der Umgebung Um die virtuellen Schatten allerdings realistisch in die reale Szene einfügen zu können, müssen Teile der Realität erkannt werden. Wenn man z. B. die Materialeigenschaften eines realen Objektes detektieren könnte, so könnten diese benutzt werden, um das Aussehen des virtuellen Schattens zu beeinflussen. Des Weiteren wäre es möglich die realen Schatten zu detektieren, damit Überschneidungen von virtuellen auf reale Schatten realistischer gestaltet werden könnten. Außerdem würde somit die Möglichkeit bestehen, die Lichtquelle bzw. die Richtung des Lichtes und die Art des Lichtes im Bild bestimmen zu können. Damit kann das virtuelle dem realen Licht mehr angepasst werden, um den Grad des Realismus zu erhöhen.

Ein weiterer wichtiger Aspekt ist die automatische Erkennung der Objekte in der Umgebung. Da diese Erkennung der Realität bisher nur in Ansätzen funktioniert, werden alle realen Objekte mit virtuellen Modellen nachgebaut. Dies erfordert eine genaue Kenntnis der realen Szene und genügend Zeit, um die Objekte zu modellieren. Außerdem müssen diese Phantomobjekte über ein Trackingverfahren in der realen Szene integriert werden. Wünschenswert wäre es daher, alle realen Objekte und ihre Position in Echtzeit erkennen, und sie für den Schattenalgorithmus benutzen zu können.

Weiterführende Arbeiten Der Ansatz funktioniert zur Zeit nur mit einer Video-See-Through Brille. In Kapitel 5.2.3 wurden die Schwierigkeiten von optischen Displays erörtert. Trotzdem wäre es wünschenswert das optische Display zu integrieren, um alle möglichen Arten der AR-Displays zu unterstützen.

Eine komplexere Art, den Ansatz zu erweitern, ist es, das gesamte Licht der Umgebung einzufangen. Somit besteht die Möglichkeit alle virtuellen Objekte und die virtuellen Schatten korrekt zu beleuchten. Mithilfe einer Fischaugenkamera oder einer Light Probe ist das Aufnehmen der Umgebung möglich. Mit einer korrekten Beleuchtung wirken sowohl die Schatten, als auch die virtuelle Objekte in ihrer realen Umgebung realistischer.

8.2.2 Anwendungsszenarien

Für den Ansatz des modifizierten Shadow Mapping Algorithmus für Mixed Reality-Umgebungen gibt es viele Einsatzmöglichkeiten. Er kann in verschiedenen Augmented Reality-Anwendungen eingesetzt werden. Die Augmented Reality im Allgemeinen kann gut benutzt werden, um Orte oder Gebäude zu erschaffen, die schon lang nicht mehr existieren, oder vielleicht erst in der Zukunft existieren werden. Damit hat der Betrachter die Möglichkeit sich virtuell an diese Orte zu begeben und sie zu besichtigen, bzw. mit oder in ihnen zu interagieren. AR kann daher gut für die Tourismusbranche genutzt werden (Abbildung 31(a)). Ein Tourist besucht beispielsweise die Ruinen der alten Stadt Pompeji, am Golf von Neapel. Durch die Augmented Reality hat er die Möglichkeit eine virtuelle Nachbildung der Stadt, wie sie zu Zeiten der Römer existierte, sehen zu können. Um diese Nachbildung realistisch wirken zu lassen, dürfen auch keine Schatten fehlen, wie sie der Ansatz dieser Diplomarbeit entwickelt hat. Dabei muss sich das virtuelle Modell nicht nur korrekt selbst verschatten, sondern es soll auch einen virtuellen Schatten auf den realen Erdboden und die gesamte reale Umgebung werfen. Der Betrachter soll das Gefühl haben, dass die Stadt genau so an dieser Stelle gestanden hat.

Ein weiteres Anwendungsszenario ist das Training für die Montage von Bauteilen. Ein Monteur bekommt virtuell gezeigt, welches Objekt an welcher Stelle eingesetzt werden muss. Dabei ist es sehr wichtig, die Abstände und die Größe der virtuellen Objekte einschätzen zu können. Beispielsweise wird ein virtueller Motor in die Karosserie eines Autos eingebaut. Ohne den virtuellen Schatten ist es für den Monteur schwer, mit dem virtuellen Motor zu interagieren und ihn einzusetzen. Wenn der Schatten des virtuellen Motors korrekt auf die reale Karosserie fällt, so fallen die Interaktionen leichter.

Schatten sind auch bei einem virtuellen Innenausstatter wichtig. Der Benutzer hat dabei die Möglichkeit, sein eigenes Zimmer mit neuen Möbeln zu bestücken. Auch hier lassen Schatten die virtuellen Objekte deutlich realistischer erscheinen und verdeutlichen ihre Position und ihre Größe. Abbildung 31(b) zeigt ein Szene bestehend aus einem realen Zimmer und einem realen Tisch, die durch virtuelle Objekte erweitert wurden. Der Betrachter hat hier die Möglichkeit sich seine Inneneinrichtung in seinem eigenen Zimmer anzuschauen, bevor er sie kauft. Allerdings fehlen in der Abbildung einige Schatten, so dass z. B. die Position und die Größe der Lampe in der realen Umgebung schwer festzustellen ist.

(a) Tourismus [50]

(b) Inneneinrichtung [40]

Abbildung 31: Einsatzgebiete von Augmented Reality

A Quellcode

Listing 1: der Shadow Mapping Vertex Shader

```
varying vec4 projCoord;

void main()
{
    vec4 realPos = gl_ModelViewMatrix * gl_Vertex;

    projCoord = gl_TextureMatrix[0] * realPos;
    gl_FrontColor = gl_Color;

    gl_Position = ftransform();
}
```

Listing 2: der Shadow Mapping Fragment Shader

```
uniform sampler2DShadow shadowMap;
varying vec4 projCoord;

void main ()
{
    const float transparency = 0.3;
    vec4 color = gl_Color;

    float shadowValue = shadow2DProj(shadowMap, projCoord).r
                + transparency;

    shadowValue = clamp(shadowValue, 0.0, 1.0);

    gl_FragColor = color * shadowValue;

}
```

Listing 3: der PCF-Shadow Mapping Fragment Shader

```
uniform sampler2DShadow shadowMap;
varying vec4 projCoord;

void main ()
{
    const float transparency = 0.3;
    vec4 color = gl_Color;

    vec3 shadowUV = projCoord.xyz / projCoord.w;
    float mapScale = 1.0 / 512.0;

    vec4 shadowColor = shadow2D(shadowMap, shadowUV);

    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3( mapScale, mapScale, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3( mapScale, -mapScale, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3( mapScale, 0, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3(-mapScale, mapScale, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3(-mapScale, -mapScale, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3(-mapScale, 0, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3(0, mapScale, 0));
    shadowColor += shadow2D(shadowMap, shadowUV.xyz
                   + vec3(0, -mapScale, 0));

    shadowColor = shadowColor / 9.0;

    shadowColor += transparency;
    shadowColor = clamp(shadowColor, 0.0, 1.0);

    gl_FragColor = color * shadowColor;
}
```

Listing 4: Fragment Shader für das Anzeigen von Schatten auf Phantomobjekten

```
uniform sampler2DShadow shadowMap;
varying vec4 projCoord;

void main ()
{
    float rValue = shadow2DProj(shadowMap, projCoord).r;

    if(rValue > 0.0 )
    {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 0.0);
    }
    else
    {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    }
}
```

B Literaturverzeichnis

Literatur

[1] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, Mark A. Livingston. Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. *Computer Graphics*, 30(Annual Conference Series):429–438, 1996.

[2] Brandon Lloyd, J. Wendt, Naga K. Govindaraju, Dinesh Manocha. CC Shadow Volumes. In *Proceedings of the 2nd EG Symposium on Rendering*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

[3] Cass W. Everitt, Mark J. Kilgard. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *CoRR*, cs.GR/0301002, 2003.

[4] Chrys Wyman, Charles Hansen. Penumbra Maps: Approximate Soft Shadows in Real-Time. In *Proceedings of the EG Symposium on Rendering*, Springer Computer Science, pages 202–207. Eurographics, Eurographics Association, 2003.

[5] Franklin C. Crow. Shadow Algorithms for Computer Graphics. In James George, editor, *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, volume 11, pages 242–248. ACM Press, July 1977.

[6] Daniel Scherzer. Real-Time Soft Shadows. In *Eurographics State of the Art Reports*. Eurographics, 2004.

[7] Eric Chan, Frédo Durand. Rendering Fake Soft Shadows with Smoothies. In *Proceedings of the EG Symposium on Rendering*, Springer Computer Science. Eurographics, Eurographics Association, 2003.

[8] Eric Chan, Frédo Durand. An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, pages 185–195. Eurographics Association, 2004.

[9] Randima Fernando. Percentage-Closer Soft Shadows. NVIDIA, 2005. `http://download.nvidia.com/developer/presentations/2005/SIGGRAPH/PCSS.pdf`.

[10] Florian Kirsch, Juergen Doellner. Real-Time Soft Shadows Using a Single Light Sample. In *Proceedings of Winter School on Computer Graphics 2003*, 2003.

[11] Fraunhofer IESE. Virtuelles Software Engineering Kompetenzzentrum. World Wide Web, Jan 2001. `http://www.software-kompetenz.de/`.

[12] Game Tutorials. Game Tutorials. World Wide Web, April 2003. `http://www.gametutorials.com`.

[13] Gabriel Gaus. Regionales Rechenzentrum für Niedersachsen, 2005. `http://www.rrzn.uni-hannover.de/`.

[14] Georg Mischler. Glossar der Lichtplanung, 1998. `http://www.schorsch.com/de/kbase/glossary/halbschatten.html`.

[15] Tim Heidmann. Real Shadows, Real Time. *Iris Universe*, 18:28–31, 1991. Silicon Graphics, Inc.

[16] J. Blinn. Me and My (Fake) Shadow. *IEEE Comput. Graph. Appl.*, 8(1):82–86, 1988.

[17] J. Sindholt. A comparison of shadow algorithms. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005. Supervised by Assoc. Prof. Niels Jørgen Christensen.

[18] Jean-François St-Amour, Pierre Poulin, Eric Paquette. Soft Shadows from Extended Light Sources with Penumbra Deep Shadow Maps. In *Proceedings of Graphics Interface*, pages 105–112, 2005.

[19] Jeff Juliano, Jeremy Sandmel. Framebuffer Object Specification. World Wide Web, April 2006. `http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt`.

[20] Joachim Böhringer, Perter Bühler, Patrick Schlaich, Hanns-Jürgen Ziegler. *Kompendium der Mediengestaltung für Druck- und Printmedien*. Springer, 2002.

[21] Jérôme Guinot. Ozone3D.net, 2002. `http://www.ozone3d.net`.

[22] Katrien Jacobs, Celine Loscos. Classification of Illumination Methods for Mixed Reality. In *Eurographics State of the Art Reports*, pages 95 – 118, Grenoble, September 2004.

[23] Katrien Jacobs, Jean-Daniel Nahmias, Cameron Angus, Alex Reche, Celine Loscos, Anthony Steed. Automatic generation of consistent shadows for augmented reality. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 113–120, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[24] Kiyoshi Kiyokawa, Yoshinori Kurata, Hiroyuki Ohno. An optical see-through display for mutual occlusion of real and virtual environments. *isar*, 00:60, 2000.

[25] Lance William . Casting Curved Shadows on Curved Surfaces. *Computer Graphics (Proceedings of SIGGRAPH '78)*, pages 270 – 274, 1978.

[26] Lengyel E. The Mechanics of Robust Stencil Shadows, October 2002. `http://www.gamasutra.com`.

[27] Lengyel E. Advanced Stencil Shadow and Penumbral Wedge Rendering. Presentation at Game Developers Conference 2005, 2005. `http://www.terathon.com/gdc\_lengyel.ppt`.

[28] Marc Stamminger, George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, pages 557–562, 2002.

[29] Mark Harris. Mark Harris' Real-Time Graphics Research, 2004. `http://www.markmark.net`.

[30] Michael Haller, Stephan Drab, Werner Hartmann. A real-time shadow approach for an augmented reality application using shadow volumes. In *VRST*, pages 56–65, 2003.

[31] Michael Wimmer, D. Scherzer, Werner Purgathofer. Light Space Perspective Shadow Maps. In *Proceedings of the 2nd EG Symposium on Rendering*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

[32] Morgan McGuire, John F. Hughes, Kevin Egan, Mark Kilgard, Cass Everitt. Fast, Practical and Robust Shadows. Technical report, NVIDIA Corporation, Austin, TX, Nov 2003.

[33] Niko Hempe. Robuste Echtzeitschatten für komplexe, dynamische Szenen. Master's thesis, Universität Koblenz, 2005.

[34] NVIDIA. NVIDIA GeForceFX 5900, 5700 and Go5700 GPUs: UltraShadow Technology. Technical report, 2003. `http://www.nvidia.com`.

[35] P. Milgram, F. Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, E77-D(1,2), December 1994.

[36] Paul Milgram, Haruo Takemura, Akira Utsumi, Fumio Kishino. Augmented Reality: A class of displays on the reality-virtuality continuum. *SPIE*, 2351, 1994.

[37] Philip Lamb, Julian Looser, Raphael Grasset, Thomas Pintaric, Uwe Woessner, Wayne Piekarski. ARToolkit. World Wide Web, Jun 2003. `http://www.hitl.washington.edu/artoolkit`.

[38] Pixar. Pixar, 1986. `http://www.pixar.com`.

[39] Pradeep Sen, Michael Cammarano, Pat Hanrahan. Shadow Silhouette Maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 2003.

[40] R. Azuma. A survey of augmented reality. In *Teleoperators and Virtual Environments 6*, pages 355–385. ACM SIGGRAPH, 1997.

[41] Randima Fernando. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. NVIDIA Corporation, 2004.

[42] Randi J. Rost. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, January 2006.

[43] S. Brabec, H. Seidel. Shadow volumes on programmable graphics hardware, 2003.

[44] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, John C. Hart. ZP+: correct Z-pass stencil shadows. In *ACM Symposium on Interactive 3D Graphics and Games*. ACM, ACM Press, April 2005.

[45] Samuli Laine. Split-Plane Shadow Volumes. In *Proceedings of Graphics Hardware 2005*, pages 23–32. Eurographics Association, 2005.

[46] Rodrigo L. S. Silva. Introduction to Augmented Reality, 2003. `http://virtual01.lncc.br/~rodrigo/links/AR/Tec-Report-AR.html`.

[47] Simon Gibson, Alan Murta. Interactive Rendering with Real-World Illumination. In *Rendering Techniques*, pages 365–376, 2000.

[48] Simon Gibson, Jonathan Cook, Toby Howard, Roger J. Hubbold. Rapid Shadow Generation in Real-World Lighting Environments. In *Rendering Techniques*, pages 219–229, 2003.

[49] Stefan Brabec, Hans-Peter Seidel. Single Sample Soft Shadows using Depth Maps. In *Proceedings of Graphics Interface*, 2002.

[50] Stefan Mueller. Computergrafik 1 und 2 und VR/AR-Folien der Arbeitsgruppe Computergrafik Koblenz. World Wide Web. `http://www.uni-koblenz.de/FB4/Institutes/ICV/AGMueller`.

[51] Steven Parker, Peter Shirley, Brian Smits. Single sample soft shadows, 1998.

[52] Timo Aila, Tomas Akenine-Möller. A Hierarchical Shadow Volume Algorithm. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*. ACM Press, 2004.

[53] Tobias Martin, Tiow-Seng Tan. Anti-aliasing and Continuity with Trapezoidal Shadow Maps. In *Proceedings of the 2nd EG Symposium on Rendering*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

[54] Tom Lokovic, Eric Veach. Deep Shadow Maps. In *Proceedings of SIGGRAPH '00, Computer Graphics* Proceedings, Annual Conference Series, pages 385–392. ACM SIGGRAPH, 2000.

[55] Tomas Akenine-Möller,Ulf Assarsson. Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges. In P. Debevec and S. Gibson, editors, *Thirteenth Eurographics Workshop on Rendering*, 2002.

[56] Ulf Assarson, Tomas Akenine-Möller. A Geometry-Based Soft Shadow Volume Algorithm Using Graphics Hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 2003.

[57] W.H. de Boer. Smooth penumbra Transitions with Shadow Maps. In *Submitted to Journal of Graphics Tools*, pages 185–195. AK Peters, 2004.

[58] William T. Reeves, David Salesin, Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (Proceedings of SIGGRAPH '87)*, pages 283–291, 1987.