UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Untersuchung von Rendertechniken in einem Remote Rendering Setup

Masterarbeit

Zur Erlangung des Grades eines Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Hans-Christian Wollert

Erstgutachter: Prof. Dr. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter: Gerrit Lochmann, M.Sc.
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Februar 2014

Erklärung

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

	Ja	Nein
Mit der Einstellung dieser 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 Masterarbeit
Hans-Christian Wollert
(Mat. Nr. 206 210 023)

Thema: Untersuchung von Rendertechniken in einem Remote Rendering Setup

Die Entwicklung ultraportabler, mobiler Computer ist in den letzten Jahren stark voran geschritten. Dennoch reicht die Rechenkapazität nicht an die stationärer Geräte heran. Um die maximale Renderqualität aktueller Computertechnologie für mobile Computer bereitzustellen, wird die Berechnung (teilweise) von einem stationären Gerät übernommen und dem mobilen Gerät über eine Netzwerkschnittstelle bereitgestellt.

Durch auftretende Latenzen kann die Benutzerinteraktion in einer 3D-Umgebung eingeschränkt werden. Remote Rendering Verfahren zielen darauf, direkte Benutzerinteraktion zu ermöglichen und die entstehenden Fehler gering zu halten.

In dieser Arbeit werden verschiedene Ansätze für Remote Rendering untersucht. Hierfür wird angenommen, dass das lokale Gerät, welches das finale Bild ausgibt, über eine bestimmte Rechenleistung verfügt. Somit lässt sich die Netzwerkkomponente als einen Puffer mit einer variablen Latenz und beschränkten Bandbreite innerhalb des Renderingprozesses ansehen.

Insbesondere sollen die Fragen beantwortet werden, wie groß die Fehler bei den Remote Rendering Verfahren sind, wie stark sich die Latenzen auf diese auswirken und wie und welche Rendertechniken sich damit verbinden lassen.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Einarbeitung in das Thema Remote Rendering
2. Implementierung verschiedener Ansätze
3. Evaluation und Vergleich der Implementationen
4. Dokumentation der Ergebnisse

Koblenz, 22.08.2013

– Hans-Christian Wollert –

– Prof. Dr. Stefan Müller –

Danksagung

Vielen Dank an:

Guido Schmidt, für die Unterstützung bei den Shadern für die Screenspace Reflections

Bastian Krayer, für die Unterstützung bei den Theorien zum Warping von Reflektionen

Kurze Zusammenfassung

Remote Rendering ist eine Möglichkeit aufwändige Grafik von leistungsstarker Hardware für leistungsschwächere Endgeräte bereitzustellen. Durch den Transfer der Daten über ein Netzwerk entsteht eine Verzögerung, welche die Interaktivität einschränkt. Ein Verfahren, um Befehle an eine virtuelle Kamera auf dem Endgerät direkt umzusetzen, wird 3D-Warping genannt. Das Verfahren erzeugt jedoch Bildartefakte. In dieser Arbeit werden verschiedene Ansätze für Remote Rendering Setups aufgezeigt. Zusätzlich werden die auftretenden Artefakte des Warpings und Methoden zur Verbesserung des Verfahrens beschrieben. Es werden eigene Implementationen und Verbesserungen vorgestellt und untersucht.

abstract

Remote rendering services offer the possibility to stream high quality images to lower powered devices. Due to the transmission of data the interactivity of applications is afflicted with a delay. A method to reduce delay of the camera manipulation on the client is called 3d-warping. This method causes artifacts. In this thesis different approaches of remote rendering setups will be shown. The artifacts and improvements of the warping method will be described. Methods to reduce the artifacts will be implemented and analyzed.

Inhalt

1 Einleitung

Mobile Geräte wie Smartphone, Tablets und Handhelds gewinnen in der heutigen Gesellschaft immer mehr an Bedeutung. Im Jahr 2013 wurden erstmals mehr Smartphones als Handys verkauft [1]. Auch zeigen Prognosen einen ähnlichen Umschwung von PCs und Notebooks zu Tablets [2]. Diese Geräte erreichen durch ihre geringe Baugröße, ihr geringes Gewicht und lange Akkulaufzeiten einen höheren Freiheitsgrad für ihre Verwendung. Neben der Nutzung von Webdiensten und Kommunikation spielt in der Branche der Mobilfunkgeräte Unterhaltung eine große Rolle. So sollen auch Videospiele eine immer realistischere Grafik bieten.

Aber auch auf etablierten Rendering-Plattformen wie Spielekonsolen oder Computern ist das Verlangen nach immer aufwändigeren Darstellungen weiterhin vorhanden. So versuchen sich Hardwareproduzenten mit neuen Grafikbeschleunigern gegenseitig zu übertrumpfen und auch Softwareanbieter bieten laufend neue Lösungen, um das Rendern von 3D Inhalten glaubwürdiger zu gestalten.

Abbildung 1: Aktuelles Smartphone Spiel[1]

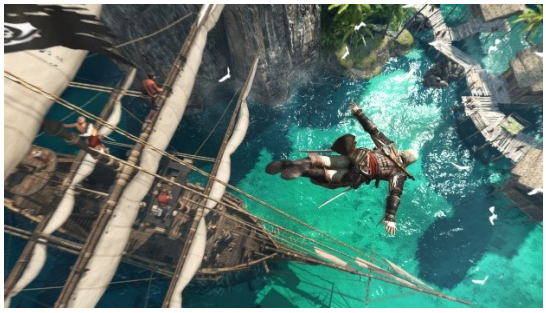


Abbildung 2: Aktuelles PC Spiel[2]

1.1 Motivation

Die grundlegende Hardwarearchitektur von Smartphones und Tablets unterscheidet sich nur wenig von der Von-Neumann-Architektur klassischer Computer, dennoch ist die Grafikleistung jener, um ein vielfaches niedriger. Dies ist unter anderem darin begründet, dass PCs über ein größeres Volumen verfügen. So besitzen sie größere Platinen und Chips, welche mehr Spannung aufnehmen können und gleichzeitig effektiver gekühlt werden können.

[1] Bildquelle: http://assassinscreed.ubi.com/de-DE/games/assassins-creed-pirates/index.aspx
[2] Bildquelle: http://assassinscreed.ubi.com/de-de/media/index.aspx

Auch die Software der vergleichsweise jungen mobilen Betriebssysteme befindet sich noch nicht auf dem gleichen Niveau der stationären Systeme. So kommen in OpenGL ES[3] nicht die umfangreichen Shader der OpenGL Rendering Pipeline oder Compute Shader zum Einsatz.

	Qualcomm Adreno 303[4]	**NVIDIA GeforceGTX 780 Ti[5]**
Chiptakt (MHz)	450/550	876
GFLOPS	158.4	5046
GP/s	3.6	40.2

Tabelle 1: Vergleich eines mobilen Chips und eines Desktopgrafikchips

Um die Leistung von Desktop PCs für mobile Endgeräte nutzen zu können ist ein effizientes Remote Rendering Verfahren notwendig. Dieses Verfahren zielt darauf ab, die aufwändigen Berechnungen auf einen Server auszulagern, die Benutzereingaben zu synchronisieren und die auftretende Verzögerung durch die Netzwerkkomponente weitestgehend zu komprimieren. Auch stationäre Geräte können durch ein Serversystem in einem Remote Rendering Setup profitieren, welches hoch komplexe grafische Aufgaben übernimmt und dem Client bereitstellt. Somit muss sich der Client auch nicht auf der benötigten technologischen Stufe befinden, um aufwändige Grafik darstellen zu können.

Ein solches Remote Rendering Verfahren könnte in allen Bereichen interessant sein, in denen eine sehr aufwändige oder realistische Grafik relevant ist. Dabei kann es sich um Computerspiele handeln, Simulationen oder Präsentationen von industriellen Prototypen. Auch wollen Unterhaltungsdienstleister einen immer größeren Kundenkreis erreichen, welcher über eine ebenso umfangreiche Anzahl unterschiedlicher Hardware verfügt. Ein Remote Rendering System kann durch die Auslagerung der Berechnung auf eigene Computer, die grafische Qualität steigern und die Abhängigkeit von der Hardware der Kunden verringern.

1.2 Zielsetzung

In dieser Arbeit liegt der Fokus auf der Untersuchung grafischer Effekte, um die Problematik der Verzögerung durch die Netzwerkkomponente in einem Remote Rendering Setup zu behandeln. Es werden verschiedene Ansätze aus bisherigen Erkenntnissen un-

[3] http://www.khronos.org/api/opengles/2_X
[4] http://en.wikipedia.org/wiki/Adreno
[5] http://de.wikipedia.org/wiki/Nvidia-Geforce-700-Serie

tersucht und bewertet. Auch werden eigene Ansätze vorgestellt. Diese werden auf die generelle Nutzbarkeit hin untersucht und Bezug zu den bisherigen Systemen genommen. Diese Ansätze können unter verschiedenen Ansprüchen und Herausforderungen, beispielsweise der Geringhaltung der Datenpakete, die über die Netzwerkkomponente verschickt werden, entwickelt werden.

1.3 Annahmen

Die Server der Remote Rendering Systeme, welche Bilder an die Clients senden, werden nicht auf ihre Leistungsfähigkeit untersucht. Es wird angenommen, dass es sich um Systeme handelt, welche immer die benötigten Daten mit mindestens 60 Bilder pro Sekunde liefern kann.

Zur weiteren Unterscheidung wird zwischen drei verschiedenen Clientsystemen differenziert. Bei der ersten Kategorie handelt es sich um Geräte der niedrigsten Leistungsstufe, wie beispielsweise Tablets und Smartphones. In der zweiten Kategorie befinden sich Geräte wie Notebooks und Office-Computer, welche einfache grafische Berechnungen durchführen können. Die letzte Stufe beinhaltet für Spiele und 3D-Rendering ausgelegte Hardware.

1.4 Zur Gliederung

Kapitel 2 soll einen Überblick zu dem generellen Aufbau eines Remote Rendering Setups geben und einige entscheidende Techniken in einem solchen System beschreiben. Verschiedene Verfahren von Remote Rendering Systemen und einige bisherige Umsetzungen aus der Forschung und Industrie werden in Kapitel 3 erläutert. In Kapitel 4 werden die Konzepte eigener Ansätze vorgestellt. Das darauf folgende Kapitel 5 enthält den Versuchsaufbau zur Evaluierung in der Praxis. Kapitel 6 fasst die aus den in dieser Arbeit gewonnenen Erkenntnissen zusammen. Zu guter Letzt werden Möglichkeiten für weitere Forschungsfragen genannt.

2 Grundlagen und Begriffe

In diesem Kapitel werden Terminologien und Technologien beschrieben, die zum weiteren Verständnis in der folgenden Arbeit notwendig sind. Dabei handelt es sich um grundlegende Beschreibungen, beispielsweise dem Prozess des Renderns von Bildern und weiteren Technologien aus diesem Bereich, sowie Beschreibungen von Netzwerkverzögerungen.

2.1 Rendering

Ein Ziel der Computergrafik ist es qualitativ hochwertige Bilder in einer echtzeitfähigen Bildrate zu rendern. Die Qualität eines gerenderten Bildes hängt von vielen Faktoren ab. Zum Einen wird die meist subjektiv wahrgenommene Qualität von äußeren Rahmenbedingungen bestimmt, wie beispielsweise der Auflösung, die meist der nativen Auflösung des Ausgabebildschirmes entspricht. Für diesen Faktor gilt, dass mit Zunahme diesem die Bildqualität steigt, da mehr Informationen von Objekten sichtbar werden und

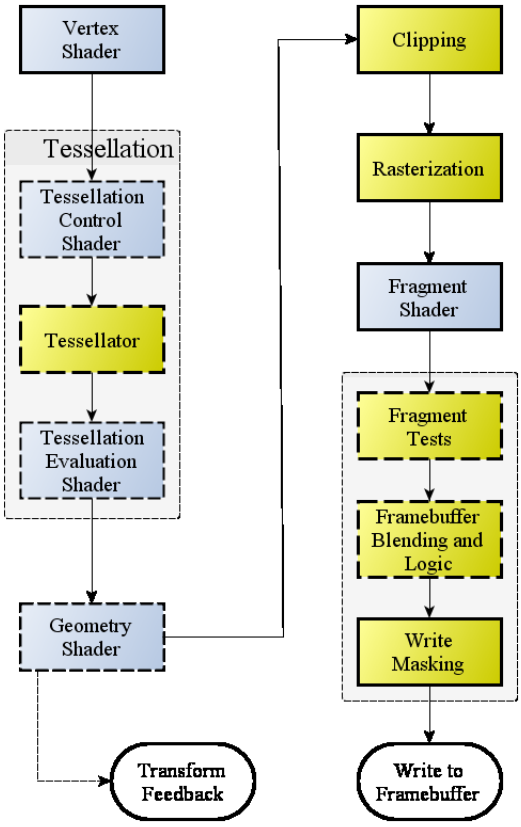


Abbildung 3: Aufbau der OpenGL Rendering Pipeline. Blaue Felder sind programmierbare, unterbrochene Rahmen sind optionale Stationen[6]

[6] Quelle: http://www.opengl.org/wiki/Rendering_Pipeline_Overview

Aliasing Effekte, also Treppenstrukturen an Objektkanten durch die Rasterung, reduziert werden.

Grundlegend für das eigentliche Bild und die damit einhergehende Qualität sind die Eingabedaten und die Verarbeitung. Die Eingabedaten entsprechen den Informationen über die virtuelle Szene, wie der Kamera und den 3D Modelldaten. Diese werden innerhalb eines Renderdurchlaufs verarbeitet. Innerhalb eines Renderdurchlaufes werden die 3D Daten anhand der Model-View-Projection-Matrix (MVP) über einen Vertexshader in Kamerakoordinaten transformiert. Die MVP besteht aus drei miteinander verrechneten Matrizen. Zum einen besteht diese aus der Model-Matrix, welche die Informationen über die Szene, beispielsweise Rotation und Skalierung der Objekte beinhaltet. Zum anderen besteht diese aus der View- und Projection-Matrix, welche die Kamerainformationen, wie Position, Blickrichtung und Öffnungswinkel beinhalten. Ein Shader ist ein Programm, welches auf der Grafikkarte(GPU) ausgeführt wird. In einem weiteren, nicht beeinflussbaren Schritt des Renderdurchgangs werden alle Daten außerhalb des sichtbaren Bereiches geclippt. Im letzten, wieder programmierbaren Schritt, können alle Daten nach der Rasterung für die Ausgabe verarbeitet werden. Hierbei werden Beleuchtungsinformationen oder andere Effekte hinzugefügt. Die Durchläufe lassen sich für jedes Bild mehrfach wiederholen um aufeinander aufbauende Effekte zu implementieren. Hierfür wird ein Framebufferobject benötigt, welches definiert, worin die Ausgabe gespeichert wird. Dabei kann es sich beispielsweise um eine oder mehrere Texturen handeln.

In dieser Arbeit wird die Bildqualität nicht durch die Qualität der Eingabedaten definiert. Es wird angenommen, dass ein vollständiger Renderdurchlauf ein optimales Bild erzeugt. Als optimal gilt in diesem Zusammenhang das gewünschte Bild, welches für jeden Bildpunkt die gewünschten Farbinformationen beinhaltet. Dies wird für die vom Server gerenderten Bilder angenommen.

Im Folgenden werden relevante Begriffe übersichtlich zusammengefasst:

- Model-, Geometriedaten (vertexdata): Menge der Grafikprimitiven (primitives), welche aus einem oder mehreren Eckpunkten (vertices) in einem 3D Koordinatensystem bestehen.
- Renderdurchlauf (renderingpass): Ausführung aller erforderlichen Schritte zur Erzeugung eines Bildes
- Clipping: Die Grafikprimitive, die sich teilweise außerhalb des Sichtkegels der Kamera befinden, werden vor der Rasterung geclippt. Der nicht sichtbare Bereich wird entfernt. Grafikprimitive, die sich komplett außerhalb befinden werden komplett entfernt.

- Weltkoordinaten (world coordinates): Die Modeldaten sind in einem einzigen, relationalen Koordinatensystem zusammengefasst. Vorab befinden sich die meisten Objekte in einem eigenen, lokalen System.
- Bildschirmkoordinaten (screenspace): Diese umfassen alle Punkte des Ausgabebereiches. Dabei kann es sich um ein Fenster oder den kompletten Bildschirm handeln. Diese Punkte beinhalten Farbewerte und optionale Werte für Transparenz (alpha).
- Kamerakoordinaten (viewspace): Alle Daten des Weltkoordinatensystems sind hier in Kamerakoordinaten transformiert. Die Kameraposition liegt im Ursprung und die sichtbaren Objekte liegen entlang der negativen z-Achse.
- (modelmatrix): Eine 4x4-Matrix, welche die Rotation, Translation und Skalierung eines jeweiligen Objektes enthält, um es in die passende Relation zu der restlichen Szene zu setzen.
- (viewmatrix): Eine 4x4-Matrix, welche die Kameraposition, Blickrichtung und den up-Vektor beinhaltet. Der up-Vektor bestimmt die y-Ausrichtung des Systems.
- (projectionmatrix): Eine 4x4-Matrix, welche die Berechnung für die perspektivische Transformation enthält.
- Rasterung (rasterization): Geometriedaten werden nach der perspektivischen Transformation in kleine Bereiche (fragments) unterteilt. Diese entsprechen den Pixeln einer Textur innerhalb des Framebuffers, der geschrieben wird.

2.2 Remote Rendering Setup

In diesem Abschnitt wird der generelle Aufbau eines Remote Rendering Systems (RRS) beschrieben. In einem klassischem 3D Programm bereitet ein Programm die Daten zur Darstellung vor und sendet diese an den 3D-Treiber im Window Manager (beispielsweise X Server in unixoiden Betriebssystemen), welcher das Ergebnis an die Ausgabegeräte sendet. In einem RRS liegt dieses Programm auf einem separaten Rechner und sendet seine Daten über eine Netzwerkschnittstelle an einen Zielrechner mit einer Ausgabe. Für die Befehle der 3D Anwendung muss in der Regel der Befehlssatz des Netzwerkprotokolls erweitert werden, beispielsweise GLX. Dennoch lastet in dem Modell der aufwendigste Teil, die Rasterung der 3D Daten, auf dem Clientsystem. Um einen Gewinn aus der Leistungsfähigkeit des Serversystems ziehen zu können, wird das Bild direkt auf dem Server erzeugt und an den Client gesendet. Der Client muss das empfangene Bild nur an die Ausgabe weiterleiten, beispielsweise VirtualGL.

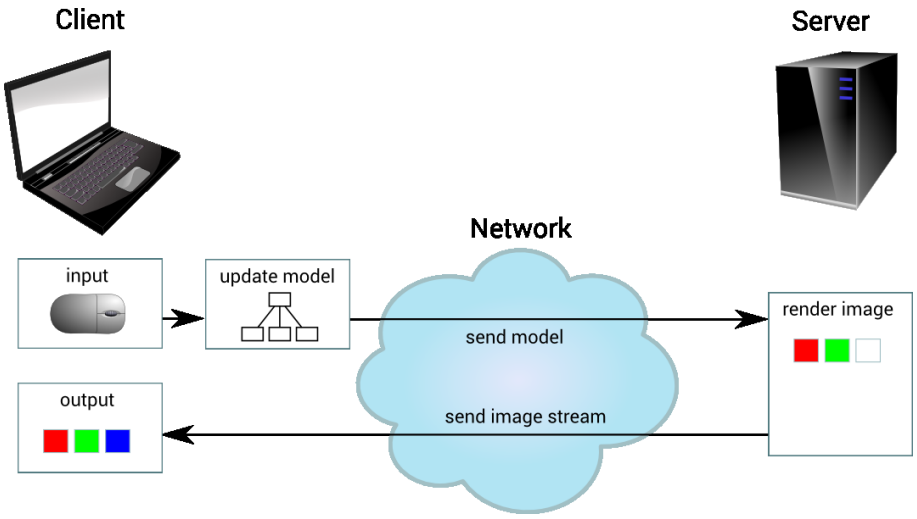


Abbildung 4: Interaktives RRS [19]

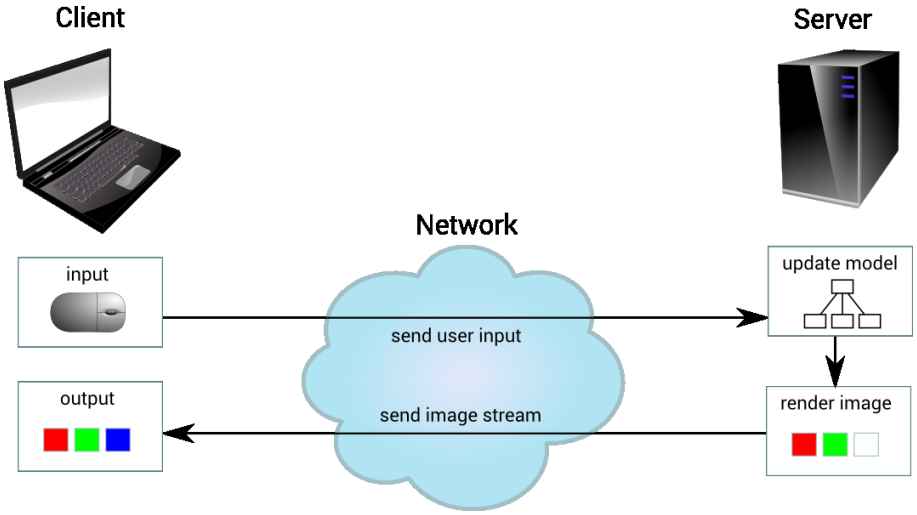


Abbildung 5: Echtzeitfähiges RRS [19]

Zusammenfassend lässt sich das Modell eines RRS auf folgender Gleichung abbilden:

$$R_i^v = render(\mathrm{S}, \mathrm{v}) \qquad (2.1)$$

$$\mathrm{I}_\mathrm{i}^{v \to v} = trans(R_i^v) \qquad (2.2)$$

$$\mathrm{I}_\mathrm{i}^{v \to v^+} = proc(\mathrm{I}_\mathrm{i}^{v \to v}, v^+) \qquad (2.3)$$

Das Referenzbild R zum Zeitpunkt i wird auf dem Server mit den 3D-Daten S und dem Viewpoint v erzeugt. Das Ausgabebild des Clients entspricht nun dem zum Zeitpunkt i übermittelten Bild. $trans()$ enthält die De- und Enkodierung des Bildes, welche ein Teil der *Latenz* ist, die in 2.3 weiter beschrieben wird. Da der Fokus in dieser Arbeit auf interaktiven 3D Szenen liegt, muss die Versendung des Blickpunktes mit hinzugezogen wer-

den. $proc()$ beschreibt die Aktualisierung des Verfahrens mit dem neuen Blickpunkt. In Abbildung 6 werden die Abläufe innerhalb des Setups näher beschrieben.

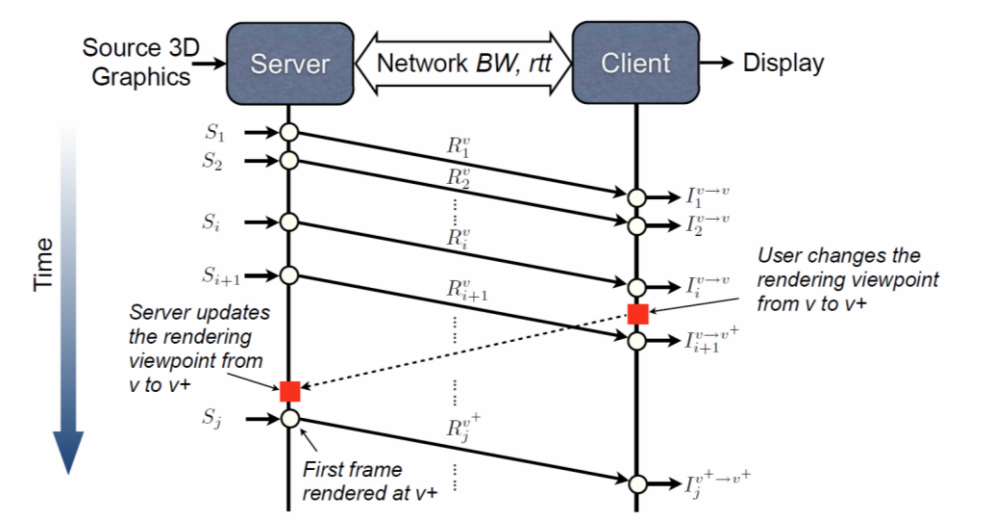


Abbildung 6: Sequenz innerhalb einer RRS [4]

Der Client stellt die vom Server gesendeten Bilder aus dem Blickpunt v so lange dar, bis der neue Blickpunkt $v+$ den Server erreicht, dieser ein neues Bild daraus rendert und an den Client sendet. In der Zwischenzeit können weitere Änderungen des Blickpunktes vorgenommen werden, die ebenso nacheinander verarbeitet und ausgegeben werden. BW (bandwidth) und rtt (round trip time) beschreiben die Eigenschaften des Netzwerkes, welche ausschlaggebend für die Verzögerung der Versendung der Bilder sind.

Um den Anspruch der Echtzeitfähigkeit zu erfüllen müssen die Renderdurchläufe in echtzeitfähigen Wiederholraten liegen:

$$\mathrm{T(render)} = \frac{1}{FPS} \qquad (2.4)$$

Die Funktion $T(x)$ beschreibt die benötigte Zeit zur Ausführung von x. FPS (frames per second) beschreibt die Bildwiederholrate. Diese kann nach Anspruch der Anwendung unterschiedlich ausfallen, liegt aber bei mindestens 30.

Ein RSS besteht aus verschiedenen einzelnen Komponenten, die wie in Abbildung 7 beschreiben zusammenhängen.

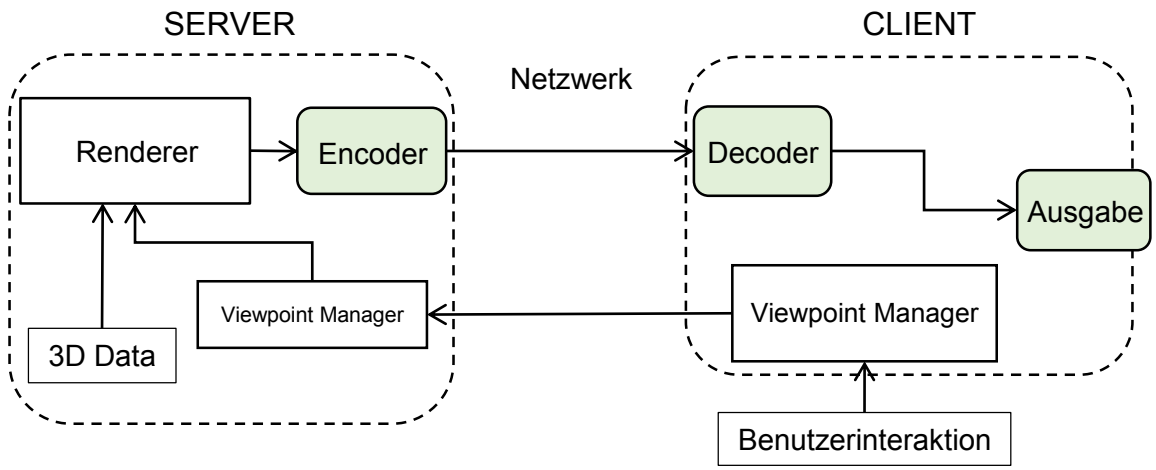


Abbildung 7: Komponenten und Aufbau eines RRS

Die Abbildung 7 illustriert das vereinfachte Model eines Remote Rendering Setup. Es beinhaltet innerhalb des Servers das Modul des Renderers, der ein Ergebnis aus den vorliegenden Daten (3D-Modell-Daten, Kamera) berechnet. In diesem einfachen Modell liegen die Daten zu Beginn der Anwendung auf einem lokal verfügbaren Speicher. Empfängt der Server einen neuen Vektor der Blickrichtung vom Client, wird diese für die nächste Berechnung anstelle der vorliegenden Blickrichtung benutzt. Der Blickpunkt umfasst die Eigenschaften der virtuellen Kamera. Dies beinhaltet zum einen die Position und die Blickrichtung der Kamera. In einer interaktiven Anwendung können diese Eigenschaften durch Benutzereingaben manipuliert werden. Hierfür ist der Viewpoint Manager zuständig. In einem weiteren Schritt wird das erzeugte Bild über ein Komprimierungsmodul (Encoder) in der Datenmenge reduziert und anschließend an den Client gesendet. Sobald der Client das Ergebnis über die Netzwerkkomponente empfängt wird dieses im Decoder sofort entschlüsselt. Wie in [3], [4] und [5] gezeigt ist dieser Schritt essenziell, um die Echtzeitfähigkeit des Systems zu garantieren. Dies ist darin begründet, das die Datenmenge nicht komprimierter Bilder die Bandbreiten gängiger Netzwerk Verbindungen übersteigen würde. In der Regel wird eine H.264 Kodierung eingesetzt, welche durch die blockweise Kodierung die Leistungsfähigkeit parallelisierter Hardware zum Beispiel der Grafikkarte nutzt [6]. Eine Verarbeitungskomponente (2D-Treiber) sendet das Bild an das Display zur Darstellung. Sobald der Benutzer eine Eingabe tätigt wird diese direkt an den Server zur weiteren Berechnung gesendet.

In der weiteren Arbeit wird davon ausgegangen, dass der Client über eine Rechenkapazität verfügt, die dazu ausreicht, das empfangene Bild weiter zu verarbeiten. Die Verarbeitung besteht in dieser Arbeit aus einem Image-Warping Prozess, der aus einem eigentlichen Warping Verfahren besteht und eventuell darauf aufbauenden Weiterverar-

beitungen. Die Funktionsweise des Warping wird in 2.4 weiter ausgeführt. Die Komponenten des einfachen RRS werden wie folgt erweitert:

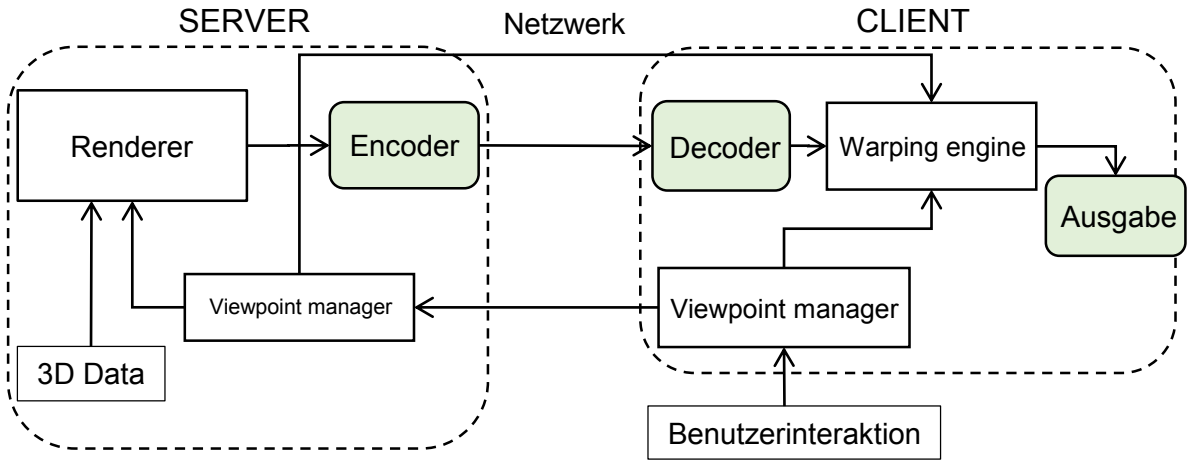


Abbildung 8: Erweitertes RRS

Die Warping Engine stellt eine zusätzliche Station innerhalb der RRS Pipeline dar, bevor das Bild an die Ausgabe gesendet wird. Diese arbeitet auf dem dekodierten Bild und sowohl dem alten, als auch dem neuen Blickpunkt.

Im Zuge der Echtzeitfähigkeit erweitert sich die Anforderung um folgende Bedingungen:

$$\mathrm{T(render)} = \frac{1}{\mathrm{FPS_s}}, \mathrm{T(warp)} = \frac{1}{\mathrm{FPS_c}} \qquad (2.5)$$

Die Größen FPS_s und FPS_c müssen nicht gleich sein. Im Idealfall ist $FPS_c \geq FPS_s$. Für den Fall, dass $FPS_c < FPS_s$, gehen vom Server gesendete Informationen verloren, da diese nicht rechtzeitig dargestellt werden können.

Unter der Voraussetzung der Leistungsfähigkeit des Clients lässt sich das Model zu einem hybriden Ansatz erweitern. So erweitert man den Client mit einem eigenen Renderpass. Das Rendering des Clients entspricht nicht der Qualität des Servers. Um das Bild auf dem Client zu rekonstruieren werden beide Bilder über ein Compositing zusammengeführt.

Insbesondere Algorithmen, die auf dem Bildbereich arbeiten, können von diesem Ansatz profitieren, da alle sichtbare Geometrie bekannt ist. Bei einem solchen Verfahren kann es sich beispielsweise um ein Beleuchtungsverfahren handeln.

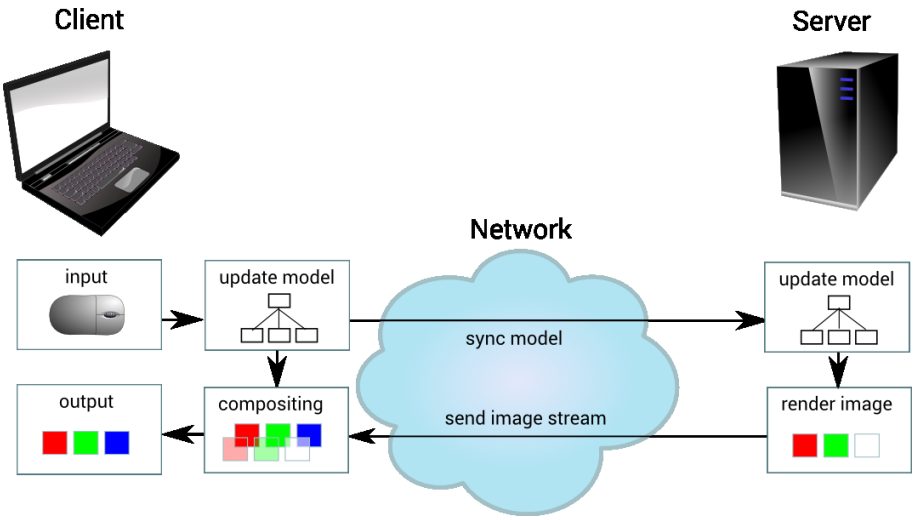


Abbildung 9: Hybrides RRS [19]

2.3 Latenz

Die Latenz beschreibt in Bezug auf RRS die komplette Zeitspanne, die zwischen der Eingabe des Benutzers und der Darstellung des auf der Eingabe basierten Bildes vergeht. Wie in Abbildung 5 veranschaulicht, entspricht dies dem Verlauf von der Benutzereingabe auf dem Client (rotes Quadrat) bis zur Darstellung von $I_j^{v^+ \to v^+}$. Bei den einzelnen Schritten handelt es sich, wie im vorherigen Kapitel erwähnt, um die Komprimierung des Bildes auf dem Server und der Dekomprimierung auf dem Client. Diese Größe wird im Zuge der Arbeit nicht weiter untersucht, da diese nicht relevant für die untersuchten Rendertechniken ist. Es wird lediglich ein hypothetischer Wert für die Größe der Latenz in den Untersuchungen angenommen. Bestimmt wird die Latenz auch von der Größe des Bildes und der vorliegenden Bandbreite im Netzwerk. Auch der Aufbau des Netzwerks, beispielsweise Traffic, Knoten und/oder geographische Lage hat Einfluss auf die Übertragungsgeschwindigkeit. Das schlägt sich in der rtt [7] nieder. Die rtt beschreibt die Zeit, die ein Datenpaket innerhalb eines Netzwerks benötigt um das Ziel zu erreichen und wieder zurück gesendet zu werden.

Es setzt sich aus folgenden Größen zusammen:

$$\mathrm{T}_{\mathrm{resp}}(\mathrm{v} \to \mathrm{v}^+) = T(warp) + T(render) + T(dec) + T(enc) + rtt + \frac{R_i^v}{BW} \qquad (2.6)$$

Die Größen $T(dec)$ und $T(enc)$ entsprechen den Zeiten der De- und Komprimierung. R_i^v entspricht der Größe des weitergeleiteten Bildes. Dessen Übertragungsgeschwindigkeit ist abhängig von der zur Verfügung stehenden Bandbreite. Ebenso wie die rtt kann diese innerhalb einer Anwendung, durch Störfaktoren verursacht, variieren. Dieses Verhalten

wird in der Arbeit nicht beachtet, da festgelegte Werte immer als höchster Auftretender Wert angenommen werden.

2.3.1 Beispiele

Das Programm Steam® In-Home Streaming[7] ist ein Streamingdienst der Spieleplatform Steam®. Der Dienst bietet einen Einblick in die Verteilung und Beanspruchung der Netzwerkkomponente. Eine weitere Beschreibung folgt in einem späteren Kapitel.

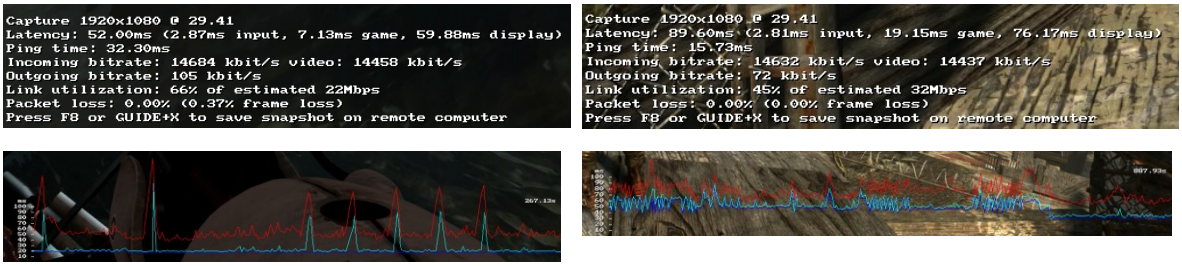


Abbildung 10: Statistiken (oben) und Verläufe (unten) der Spiele Portal2 (links) und Tomb Raider (rechts)

Die Abbildungen zeigen die Statistiken und den zeitlichen Verlauf zweier Programme. Beide Rechner sind mit dem lokalen Netzwerk über WLAN verbunden. Erfasst werden hier unter anderem die verschiedenen Verzögerungen von der Eingabe, der Berechnung des Bildes und der Zusendung. Die rtt ist hier als ping time angegeben. Die Latenzen schwanken zwischen ca. 50 und 150 Millisekunden. Mit diesen Verzögerungen ließen sich die Figuren der Programme problemlos kontrollieren. Die Netzwerkauslastung liegt bei der weitverbreiteten Auflösung von 1920 x 1080 bei 14Mbit/s. Das lokale WLAN wurde damit nur zwischen 40 und 60 Prozent ausgelastet.

2.4 3D Warping

Warping (engl. Krümmung, Verziehen) beschreibt den Vorgang des Überführens oder Zerrens von Bildpunkten von einem Bekannten in ein neues Kamerasystem. In der Computergrafik entspricht das dem Erzeugen eines abgeleiteten Bildes aus einem Referenzbild. Das Verfahren findet unter anderem Anwendung darin Bildraten (beispielsweise eines Videos) zu erhöhen und damit die Übergänge zwischen den einzelnen Bildern zu glätten. Aber auch in dem in 2.2 angesprochenen Setup lässt sich das Verfahren anwenden, um die auftretenden Verzögerungen bei der Interaktion des Benutzers zu vermeiden.

[7] http://steamcommunity.com/groups/homestream?l=german

Die ersten Ansätze des Warpings dienten dazu, durch Bildverschiebung Latenzen zu reduzieren. Das ist eine reine 2D Lösung und wurde bald um die Rotation und Skalierung erweitert. Um gezielter zufällige Kamerabewegungen behandeln zu können wurde eine perspektivische Transformation der Bilder hinzugezogen. Unter der Berücksichtigung von Tiefenwerten wurde das Warping zur Blickpunkt Interpolation genutzt und findet ebenso Verwendung in stereoskopischen Displays [8].

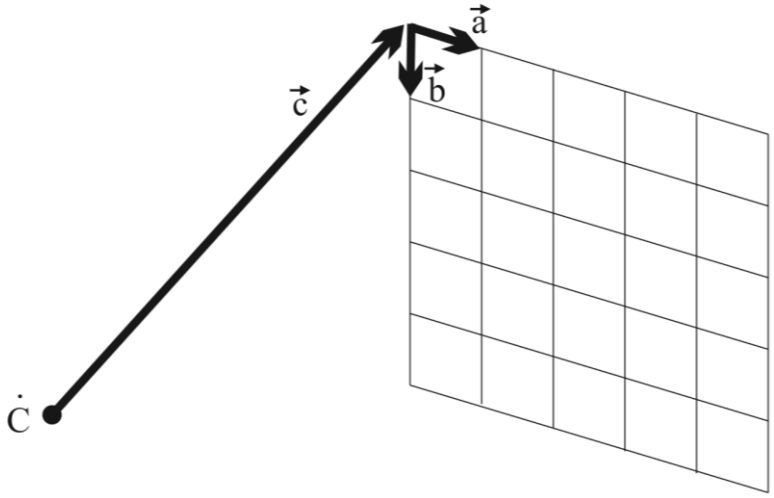


Abbildung 11: Auflösungsabhängiges, planar-projiziertes Kameramodell [8]

Innerhalb des planar-projektivischen Kameramodells entspricht nach Mark [9] ein Punkt im abgeleiteten Bild dem korrespondieren Punkt im Referenzbild. Hierbei gilt zu beachten, dass es sich um ein auflösungsabhängiges Modell handelt, bei dem die berechneten Bildkoordinaten $(u, v)^T$ nicht im Wertebereich von [0,1] liegen, sondern im Bereich $[0, Bildweite]$ und $[0, Bildhöhe]$ (siehe Abbildung 11).

Für die korrespondierenden Punkte $u_2 = (u_2, v_2, 1)^T$ und $u_1 = (u_1, v_1, 1)^T$ ergibt sich folgende Gleichung:

$$\dot{C}_2 + \frac{R_2}{S_2} z_2 u_2 = \dot{C}_1 + \frac{R_1}{S_1} z_1 u_1 \qquad (2.7)$$

$\dot{C}$ ist, wie in Abbildung 11 gezeigt, die Position der Kamera. R ist eine 3x3-Matrix mit den intrinsischen Kameraparametern und der Blickrichtung. S ist ein Skalierungsfaktor zwischen der Bildebene und der Kameraposition, der sich wie folgt berechnet:

$$S = \frac{\vec{a} \times \vec{b}}{\|\vec{a} \times \vec{b}\|} \cdot \vec{c} \qquad (2.8)$$

Die Tiefenwerte z entsprechen den standardisierten Werten des Tiefenpuffers:

$$z = \vec{p} \cdot \vec{n} \qquad (2.9)$$

$\vec{p}$ ist ein Vektor zwischen dem 3D Punkt und der Kamera. $\vec{n}$ ist der normalisierte Blickvektor. Zur Berechnung des abgebildeten Punktes ergibt sich aus 2.7 folgende Gleichung:

$$\frac{z_2}{S_2}u_2 = R_2^{-1}(\dot{C}_1 - \dot{C}_2) + \frac{R_2^{-1}R_1}{S_1}z_1u_1 \quad (2.10)$$

Hierbei lässt sich der erste Term der rechten Seite als Translationsanteil des Warpings betrachten. Dies entspricht der Verschiebung von der einen Kamera- in die neue Kamerablickrichtung. Der zweite Term entspricht der eigentlichen Überführung des Punktes in das neue Kamerakoordinatensystem.

Zusammenfassend lässt sich der Prozess des Warpings auf die Schritte des Deprojizierens, einer Rotation, einer Translation und dem erneuten Projizieren unterteilen. So lassen sich diese Schritte auch auf die Rendering Pipeline übertragen. Der Punkt in Weltkoordinaten $p_w = (x, y, z, 1)^T$ entspricht einem Punkt im Referenzbild $p_1 = (x_1, y_1, z_1, 1)^T$ und im abgeleiteten Bild $p_2 = (x_2, y_2, z_2, 1)^T$ ohne ihre perspektivische Transformation.

$$p_w = MVP_2^{-1} * p_2 = MVP_1^{-1} * p_1 \quad (2.11)$$

Zur Berechnung des abgeleiteten Punktes ergibt sich analog:

$$p_2 = MVP_2 * MVP_1^{-1} * p_1 \quad (2.12)$$

Die MVP ist die Model-View-Projection-Matrix, mit der das jeweilige Bild erzeugt wurde. Sie enthält sowohl die intrinsischen Kameraparameter und damit die perspektivische Verzerrung in der Projection-Matrix, als auch die Translation in der View-Matrix. Die model-Matrix enthält zusätzliche Informationen über Rotation, Skalierung und Translation über die Geometrie, welche benötigt wird, um die exakten Weltkoordinaten zu bestimmen. Da der Punkt p_1 vor dem Warping in Bildkoordinaten vorliegt, muss dieser vorab in den Wertebereich $[-1, 1]$ gebracht werden. Dies entspricht den Normalized Device Coordinates. Der abgeleitete Punkt p_2 muss analog in Bildkoordinaten zurückgerechnet werden.

2.5 **Reflektionen**

Reflektionen, oder auch Spiegelungen, kommen in der realen Welt sehr häufig vor, beispielsweise auf Wasseroberflächen, Gläsern oder im klassischen Spiegel. Generell treten Spiegelungen auf sehr glatten Oberfläche auf. Um solche Oberflächen darzustellen müssen Reflektionen innerhalb der 3D Szene simuliert werden.

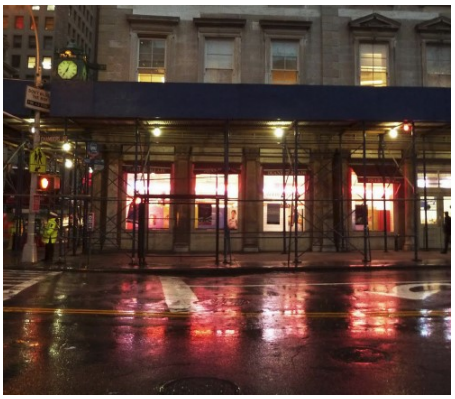


Abbildung 12: Reflektionen in der Wirklichkeit [10]

Ein einfaches Modell des Lichts beschreibt es als eine Menge von Photonen, kleine Energiepartikel innerhalb einer Wellenlänge, die mit Lichtgeschwindigkeit durch den Raum fliegen und von der Netzhaut des Auges aufgenommen werden. Die entstehenden Reize im Auge werden im Gehirn zu einem Bild zusammengesetzt. Trifft ein Photon auf eine Oberfläche, verliert es Energie und ändert seine Wellenlänge. Diese Veränderung wird im Auge als Farbe wahrgenommen. In einem perfekten Spiegel wird das Licht ohne Energieverlust reflektiert. Das eigentliche Objekt ist nicht zu sehen, sondern nur alle darin sich spiegelnde Objekte. Dies tritt in der Realität nie ein, da jedes Objekt eine gewisse Energieabsorption verursacht. So ist immer ein Anteil des eigentlichen Objektes sichtbar. Zusätzlich sind Oberflächen nicht komplett plan und gespiegelte Objekte sehen meist nicht so aus, als würde man sie direkt betrachten. Beispielsweise werden Verzerrungen durch Wölbungen und Dellen hervorgerufen oder das Abbild wird durch Unebenheiten unscharf und unterbrochen. Bei letzterem wird auch von nicht perfekten Reflektionen (glossy reflections) gesprochen.

Um Reflektionen in der Computergrafik zu approximieren gibt es mehrere unterschiedliche Ansätze. Ein trivialer Ansatz dupliziert die Geometrie und transformiert sie auf die gegenüberliegende Seite der Oberfläche des reflektierenden Objekts und rendert diese zusätzlich. Die transformierte Geometrie wird an den Rändern des Spiegels geclippt. Beide Renderings werden am Ende miteinander verrechnet. Das Verfahren funktioniert für planare Spiegeloberflächen. Es muss jedoch für gekrümmte Oberfläche um ein Geometrietransformationsverfahren erweitert werden. In der Praxis findet diese Technik auf Grund der Einschränkungen kaum noch Verwendung.

Ein bildbasierter Ansatz erzeugt eine Karte der Umgebung (environment map) für das spiegelnde Objekt. Dafür wird vorab ein Punkt definiert, beispielsweise dem Schwerpunkt des Objektes. Aus diesem Punkt wird entlang der positiven und negativen Achsen

des Koordinatensystems die Szene in quadratische Texturen gerendert. Zusammengefasst lässt sich aus diesen Bildern ein Würfel (cube map) um das Objekt legen. Die reflektierten Vektoren von der Kamera auf der Oberfläche des Objektes werden benutzt, um die Cube Map abzutasten. Die Vorteile des Verfahrens liegen dadurch, dass die Cube Maps vorab berechnet werden können und das Abtasten im Rendervorgang sehr schnell ist. Nachteile ergeben sich darin, dass die Reflektionen nicht genau sind, da die entsprechenden Reflektionspunkte dem definierten Punkt entsprechen, aus dem die Environment Map generiert wurde. Ein weiterer Nachteil besteht darin, dass bei Änderungen der Szene oder einer Verschiebung des Objekts die Environment Map neu erstellt werden muss.

Eine Methode, komplette Reflektionen auch über mehrere Oberflächen zu approximieren, basiert auf einem anderen Renderingverfahren gegenüber dem herkömmlichen Scanline Rendering aus 2.1. Das Raytracing genannte Verfahren basiert auf der Simulation von Strahlen, die aus der Kamera in die Szene geschossen und mit der Geometrie auf Schnittpunkte untersucht werden. Handelt es sich bei einem gefunden Schnittpunkt um eine diffuse Oberfläche, wird dieser Wert in den Pixel eingetragen. Handelt es sich um eine spiegelnde Oberfläche, wird der Reflektionsvektor weiterverfolgt und auf weitere Schnittpunkte untersucht. Das kann für andere, spiegelnde Oberflächen weitergeführt werden. Die Rekursionstiefe sollte von der Anwendung begrenzt werden, um im Falle von unendlichen Reflektionen zu terminieren. Der Nachteil des Raytracing Renderings liegt darin, dass die Berechnung der Schnittpunkte für die vielen Strahlen, die notwendig sind (mindestens einer pro Pixel), um ein vollständiges Bild zu berechnen, sehr aufwändig ist. Zusätzlich ist bisherige Hardware auf das Scanline Rendering optimiert. Hierarchierungsalgorithmen der Geometrie und neue Softwarelösungen zur Nutzung der Grafikkarte für General Purpose Aufgaben erlauben mittlerweile die Verwendung von Raytracing in Echtzeitanwendungen.

Ist zusätzlich zu einem gerenderten Bild die korrespondierende Tiefe verfügbar, lässt sich auch ein Raytracing innerhalb des Bildes anwenden und das Verfahren somit beschleunigen. Bei dem Screenspace Reflection (SSR) genannten Verfahren werden Reflektionen über den Bildbereich bestimmt [10]. Um den Reflektionsvektor zu berechnen werden die Positionen und Normalen der Punkte im Viewspace hinzugezogen. Dieser Vektor wird in den Screenspace umgerechnet. Schrittweise wird nun das Bild mit dem Vektor Pixel für Pixel abgetastet. Sobald der Tiefenwert eines abgetasteten Punktes grö

ßer dem Tiefenwert des Vektors ist, ist ein Schnittpunkt und der zu spiegelnde Punkt gefunden. Ein entscheidender Nachteil dieser Methode ist, dass Objekte außerhalb des sichtbaren Bereiches nicht gespiegelt werden. Rückseiten von Objekten oder Objekte hinter der Kamera sind in der Reflektion nicht vorhanden.

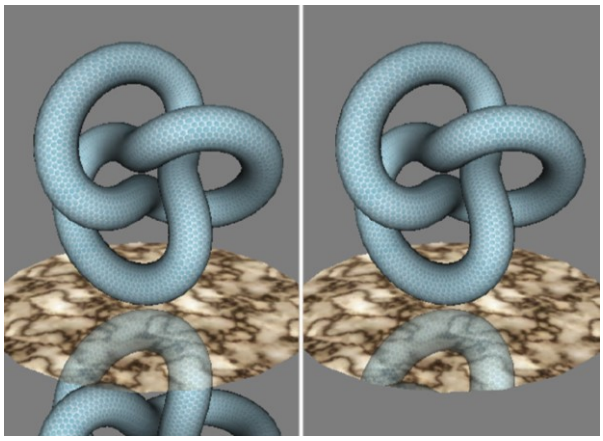


Abbildung 13: Reflektierte Geometrie [10]

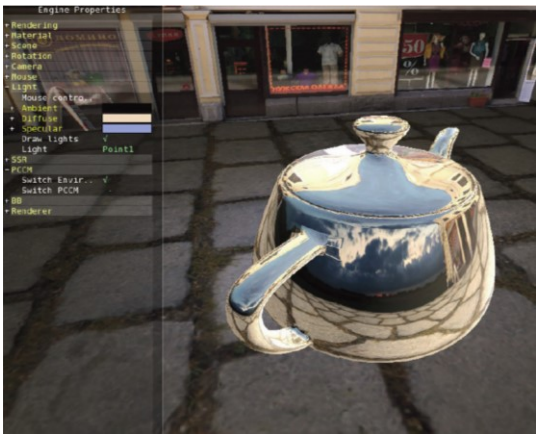


Abbildung 14: Cupe Map [10]

Abbildung 15: Raytracing [10]

Abbildung 16: Screenspace Reflections [10]

2.6 Layered reflections

Das Verfahren von Sinha et al. [11] ist kein direkter Anwendungsfall eines RSS, beschäftigt sich aber mit einer Problematik, die ebenso auf das Warping übertragbar ist. In dieser Arbeit geht es um die Untersuchung eines Image-based Rendering Systems und dem darin auftretenden Problem, dass Reflektionen innerhalb der Bilder nicht flüssig dargestellt werden können.

Die Idee des Verfahrens basiert darauf, die Szene in zwei verschiedene Ebenen mit dem diffusen und dem reflektiven Anteil zu teilen und für beide Zwischenbilder zu generieren, um die Bildrate zu glätten. Für die Ebenen werden über Korrelationsverfahren die Tiefenwerte bestimmt. Für die Glättung der Bildraten können beide Ebenen unabhängig

voneinander innerhalb einer Viewpoint Interpolation verzerrt werden. Ein letzter Schritt fügt beide Bildebenen zusammen und generiert das finale Bild.

3 Bisherige Systeme

Für RRS gibt es verschiedene Abstraktionsstufen, welche sich durch ihrer Daten und deren Menge, die über das Netzwerk gesendet werden, unterscheiden. Mit den Daten unterscheiden sich auch die Berechnungen und Belastungen des Clients.

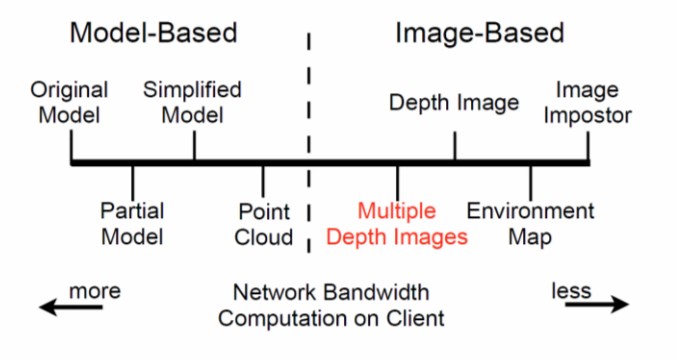


Abbildung 17: Remote Rendering Systeme [20]

Das Kapitel soll einen Überblick über verschiedene Ansätze und einige eingesetzte Systeme geben.

3.1 Modellbasierte Systeme

Modellbasierte Verfahren sind keine direkten Ansätze für ein Remote Rendering System. Der Server allein stellt nur 3D-Daten für den Client bereit. Das komplette Rendering wird vom Client durchgeführt, wie es bei GLX der Fall ist. GLX [12] ist eine Erweiterung zum X11 Protokoll, um OpenGL Befehle an einen X-Server, dem Darstellungssystem, zu senden. Eine weitere Komponente interpretiert die eintreffenden Befehle und leitet diese an die Grafikhardware weiter. Im Vergleich zum direkten Rendering ist dieses Verfahren signifikant langsamer. Ein solches System wird beispielsweise dann verwendet, wenn die gerenderten Bilder nicht in Echtzeit zur Verfügung stehen müssen und der Client durch das Rendering nicht blockiert werden soll. Zusätzlich beansprucht dieses Verfahren im Vergleich zu allen anderen Remote Rendering Verfahren am meisten Bandbreite des Netzwerks durch große Modeldaten und eventuellen Texturen. Ein solches System stellt Autodesk® 360 Rendering[8] dar, welches es erlaubt CAD Projekte über die Cloud von Autodesk® rendern zu lassen. Für jeden Renderingauftrag müssen alle neuen oder veränderten Geometrien in die Cloud hochgeladen werden. Die finalen Bilder werden über eine benutzerspezifische Schnittstelle bereitgestellt. Für interaktive

[8] http://www.autodesk.de/products/rendering/overview

Anwendungen hat sich gezeigt, dass insbesondere bei Änderungen der Szene und dem Nachladen von Objekten große Schwankungen in der Belastung der Netzwerkschnittstelle entstehen [5]. Eine aufwändige Komprimierung der Grafikbefehle und vorausschauende Bereitstellung von Daten ist in diesem Fall notwendig, da diese mit der Komplexität der Anwendung steigen.

Alternative Techniken bieten Lösungen, um nur einen Teil des originalen Modells zu senden. Dies soll die Netzwerkkomponente entlasten, sodass dem Client die Daten schneller zur Verfügung stehen. Allerdings kann sich die reduzierte Geometrie negativ auf die Bildqualität oder die Interaktivität auswirken.

Ein anderes Verfahren nach Levoy [13] erstellt Serverseitig ein vereinfachtes Model der Geometrie und erstellt mit dem originalen und vereinfachten Model zwei Renderings. Das aus den beiden Bildern erzeugte Differenzbild wird an den Client geschickt, um das originale Bild zu rekonstruieren. Zur Rekonstruktion benötigt der Client das reduzierte Modell, das optional vom Server gesendet wird oder bereits auf dem Client vorhanden ist. Dies erfordert im Vergleich zu den beiden vorangestandenen Verfahren deutlich weniger Bandbreite der Netzwerkanbindung. Dem gegenüber erfordert das Verfahren einen großen Berechnungsaufwand des Servers zum Erstellen der Daten. Der Client kann das Bild nicht aus veränderter Blickrichtung rekonstruieren.

Als weitere Alternative lässt sich die Geometrie zu einer Punktewolke umwandeln [14]. Diese wird der Auflösung des Bildschirms des Client angepasst, um die kleinste notwendige Datenmenge zu erzeugen. Das Generieren der Datenmenge ist im Vergleich zum klassischen Rendering ein Mehraufwand auf Seiten des Servers. Durch die geringe Datenmenge ist dieses Verfahren der modellbasierten Systeme das geeignetste aus Sicht der Netzwerkanforderungen. Allerdings hängt die Bildqualität stark von der erzeugten Punktewolke ab. Mit Hilfe von Splats kann die Bildqualität nachträglich verbessert werden.

3.2 Image Imposter

Der Image Imposter ist der einfachste Aufbau eines RRS, das schon in 2.2 vorgestellt wurde. Er besitzt server- und clientseitig keinerlei Nachteile durch zusätzliche Berechnung. Jegliche Information wird im Bild gespeichert und direkt ausgegeben. Somit entspricht die Bildqualität immer dem optimalen Ergebnis, das vom Server erzeugt wird. Der entscheidende Nachteil eines Image Imposter RRS liegt darin, dass Benutzereingaben um die komplette Latenz verzögert werden. Dieses Problem kann die Bedienbarkeit des Programmes stark einschränken. Das Ziel des Verfahrens ist es einen Komprimie-

rungsalgorithmus zu integrieren, der geringe Auswirkungen auf die Latenz hat. Die Algorithmen erreichen dies durch eine geringe Ausführungszeit und eine Reduzierung der benötigen Bandbreite. Durch das Netzwerk bedingte Latenzen lassen sich nicht beheben. Zusätzlich darf die Komprimierung keinen Einfluss auf die Bildqualität haben. Ein weiterer Vorteil des Verfahrens ist, dass es für fast alle Zielgeräte geeignet ist. Mit Ausnahme der Dekodierung des eingehenden Bildes wird kein Berechnungsaufwand benötigt.

Basierend auf dem Konzept des Image Imposters wurden bereits einige Projekte, auch im kommerziellen Bereich entwickelt. Als eines der kommerziellen Projekte ist der amerikanische Streamingdienst OnLive® zu nennen. Dieser Dienst bietet die Benutzung von Videospielen an, die über das eigene Cloudsystem gerendert und gestreamt werden. Allerdings ist dieser Dienst lokal begrenzt und stellt einige Anforderungen an die Internetanbindung des Kunden.

Ein weiterer Streaming Dienst ist das in Kapitel 2.3.1 bereits erwähnte Steam® In-Home Streaming. Das ist ein Dienst, der es erlaubt, die Darstellung von denen auf der Plattform erworbenen Spiele auf zwei in einem lokalen Netzwerk verbundenen Rechnern zu verteilen. Sobald der Client den Dienst startet wird das Spiel auf dem Server gestartet. Der Client bekommt das Ausgabebild gesendet. Die Qualität hängt von den auf dem Server getroffenen Einstellungen ab, von welchen auch die Bildrate abhängig ist. Bei einer Überlastung des Netzwerkes werden die gesendeten Bilder in der Auflösung reduziert. Eingaben auf beiden Systemen beeinflussen das Spiel.

Abbildung 18: Spielszene aus Portal 2 mit Steam In-Home Streaming

Abbildung 19: Spielszene aus Tomb Raider mit Steam InHome Streaming

3.3 Environment Maps

Um die Probleme der Interaktion zu beheben setzt ein erweitertes Verfahren zu den Image Impostern darauf, anstelle des Bilds, die Environment Map für die aktuelle Position des Benutzers zu senden. Somit kann der Benutzer in Echtzeit die Rotation der Kamera verändern. Das gilt nicht für die Position des Benutzers. Wird diese geändert muss der Client warten bis der Server eine neue Karte sendet. Zusätzlich beansprucht das Verfahren, durch die vielfach größeren Bilder, die Netzwerkanbindung stärker als ein Image Imposter.

Eine vergleichbare Anwendung bietet der Internetkonzern Google® in seinem Straßenbilderdienst StreetView®[9] an. In dem Dienst lassen sich Panoramaaufnahmen von Straßen ansehen, die allerdings nicht von einem Server gerendert, sondern mit speziellen Kameras aufgenommen wurden.

3.4 Tiefenbilder

Als eine alternative Erweiterung des Image Imposter Verfahrens gilt die Verwendung von Tiefenbildern. Der Client erhält neben dem Referenzbild zusätzlich eine Tiefenkarte. Die Tiefenwerte können dazu genutzt werden ein 3D Image Warping, wie in 2.4 vorgestellt, auf dem empfangen Bild durchzuführen. In den Anfängen wurde das Warping genutzt, um die Bildraten einer Ausgabe zu erhöhen. In einem RRS eignet sich das Ver-

[9] http://www.google.com/intl/en_us/maps/about/behind-the-scenes/streetview/

fahren, um die Punkte im Bildbereich auf die Änderungen des Blickwinkels durch die Benutzereingaben abzubilden. Die Beanspruchung der Bandbreite liegt nur minimal über der des Image Imposters, da nur ein zusätzlicher Wert pro Bildpunkt mitgesendet werden muss. Allerdings entstehen durch das Warping Artefakte, die im folgenden Text näher erläutert werden.

Mark et al. [15] hat gezeigt, dass es zwei unterschiedliche Methoden gibt, die Bildpunkte des Referenzbildes in das abgeleitete Bild umzurechnen. Ausgangspunkt für beide Verfahren ist das gerenderte Bild, die dazugehörige Blickrichtung und die vom Benutzer bestimmte Blickrichtung:

1. Der offensichtlichere Ansatz der Rekonstruktion ist das Abbilden jedes individuellen Pixels. Jeder einzelne Bildpunkt wird individuell, analog zu dem Verfahren in 2.4, in einem Ausgabebild abgebildet.
   Bei diesem Verfahren entstehen durch undersampling Löcher in Oberflächen, die durch Rotation oder Bewegung näher an die Kamera kommen (Abbildung 20, rechts). Diese können mit einem Interpolationsverfahren oder dem Schreiben von Splats reduziert werden. Ein weiterer Typ von Artefakten sind Aufdeckungsprobleme (Abbildung 20, links). Diese entstehen, wenn vordergründige Objekte vor einem Hintergrund an eine neue Position abgebildet werden. An der vorherigen Position besteht ein Bereich ohne Information.

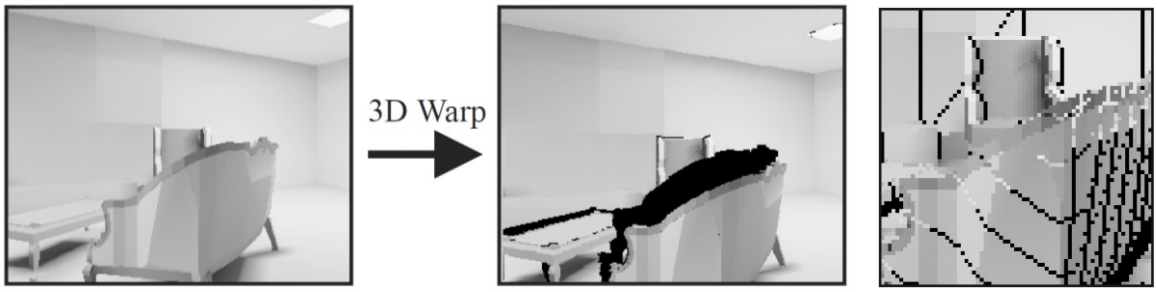


Abbildung 20: Pixelbasiertes Warping [15]

2. Der zweite Ansatz behandelt das Referenzbild als ein Mesh Objekt. Jeder Pixel entspricht einem Vertexpunkt. Die Koordinaten der Vertexpunkte entsprechen den Bildkoordinaten und der Tiefe. Diese Punkte werden analog zum anderen Verfahren gewarped, allerdings kann ein Fragmentprogramm zwischen den Punkten interpolieren. Es entstehen keine Löcher. Die neuen Bildpunkte entsprechen den x- und y-Koordinaten der gewarpten Vertexpunkte.

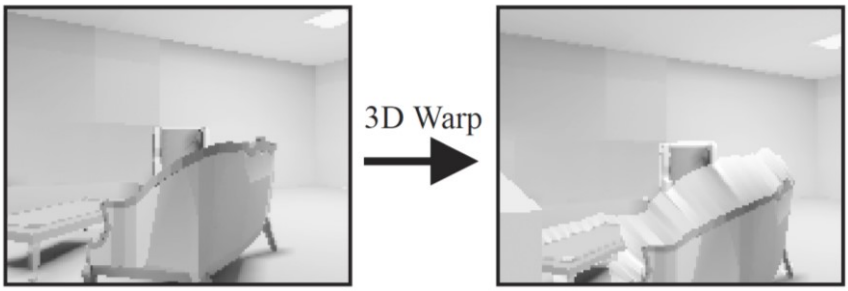


Abbildung 21: Meshbasiertes Warping [15]

Für die Minimierung der auftretenden Löcher beschreibt [9] eine einfache Lösung. Ein zusätzlicher Filter traversiert das Bild entlang des Verschiebungsvektors und schreibt in die leeren Pixel den letzten Pixel des Hintergrundbildes. Um dieses Verfahren zu optimieren wird das Bild nicht entlang des Verschiebungsvektors traversiert, sondern orthogonal zu diesem, um beim Erreichen eines leeren Pixels im Nachbarpixel entlang des negativen Verschiebungsvektors den gesuchten Farbwert zu finden. Bei konkaven Vordergrundobjekten wird ein zusätzlicher Test hinzugefügt, der zwischen Vorder- und

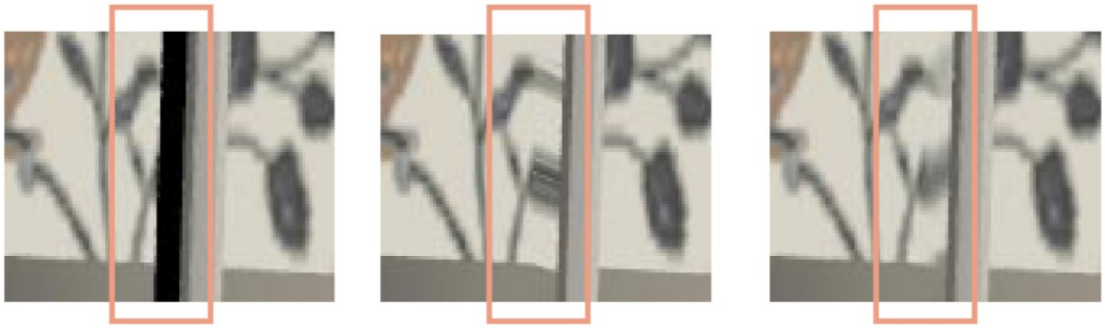


Abbildung 22: Holefilling, links ohne, Mitte ohne Weichzeichner, rechts mit Weichzeichner

Hintergrundpixel unterscheidet. Insgesamt werden die leeren Pixel mit einem streifenartigen Muster gefüllt. Ein Glättungsfilter minimiert die Auffälligkeiten dieser Bereiche.

3.5 **Multiple Tiefenbilder**

Beim Warping eines einfachen Tiefenbildes entstehen, wie bereits erwähnt, verschiedene Artefakte. Dieses Verfahren zielt darauf ab, das Sampling zu erhöhen und somit die Artefakte zu minimieren. Dafür werden vom Server mehrere Bilder aus sinnvoll bestimmten Blickpunkten gerendert und an den Client gesendet. Der Client warpt nun alle empfangenen Bilder mit dem Blickpunkt des Benutzers. Über ein Compositing[10] Verfahren

[10] Engl. Zusammensetzung

werden nun die Informationen aus allen Bildern gewichtet und entschieden, welche Informationen in das Ausgabebild geschrieben werden. Vorteile sind natürlich die (fast) optimale Bildqualität. Allerdings steigt der Berechnungsaufwand entsprechend der zusätzlich aufgewendeten Renderingdurchgänge. Auch die Netzwerkkomponente wird über das Vielfache der Referenzbilder belastet.

Für den Ansatz 1 aus 3.4 können die aus den verschiedenen Referenzbildern stammenden Abbildungen pixelweise verglichen werden. Sollten alle Abbildungen für eine Bildkoordinate einen Wert enthalten, sollte dieser für alle gleich sein. Sind die Werte unterschiedlich, wird der Wert mit der geringsten Tiefe geschrieben. Enthält eine Abbildung keinen Wert an der Stelle, wird ein Wert aus einer anderen Abbildung geschrieben. Sind alle Abbildungen an einer Koordinate leer, wird diese nicht beschrieben und bleibt leer. Im Falle des Mesh-basierten Verfahrens aus 3.4 muss dieses Compositing um zwei weitere Vergleichswerte erweitert werden [15]. Ein Wert wird bestimmt, der eine Aussage darüber geben soll, ob ein Pixel zu einer Oberfläche eines Objektes gehört oder ob es sich um ein Pixel im „rubber sheet“ beziehungsweise um einen leeren Pixel handelt. Es werden Normalen für die Vertices des Bildes erstellt und mittels eines Vorhersageverfahrens mit dem aktuellen Wert verglichen. Ein Confidence Wert gibt an, wie sich eine Oberfläche durch das Warping verändert hat und gegebenenfalls undersampled ist. Generell werden beim Compositing die Punkte mit der niedrigeren Tiefe gespeichert. Ist das nicht ausreichend, entscheidet der Connectedness Wert, ob ein Pixel zu einer Fläche gehört und deswegen geschrieben wird oder ob dieser durch einen hohen Confidence Wert zuverlässiger erscheint.

Eine zusätzliche Anforderung, die bei der Verwendung mehreren Tiefenbildern von Echtzeit Anwendungen auftritt, ist die Bestimmung weiterer Blickpunkte. Diese Blickpunkte müssen so gewählt werden, dass bei der Zusammensetzung der beiden verzerrten Referenzbilder jeder Pixel im Zielbild abgebildet wird. Folglich darf der vom Server bestimmte Blickpunkt nicht zu weit vom Betrachter entfernt sein, um Lücken zwischen den Bildern zu vermeiden. Wird der neue Blickpunkt zu nah gewählt, können Informationen am Rand des verzerrten Bildes fehlen. Der optimale Abstand hängt von den Restriktionen der Anwendungen zur Kamerasteuerung ab. Diese ist in der Regel auf eine maximale Bewegung begrenzt.

Zur Erzeugung von Referenzbildern ist es ausreichend nur diese zu erzeugen, wenn sich die Kamera bewegt. Für eine einfache Kamerabewegung in einer statischen Szene kann der Server die neue Kameraposition vorausberechnen und das Referenzbild aus dieser Position bestimmen. Für den Fall, dass der Client die Bewegung innerhalb der Latenz

verändern sollte und der vorausberechnete Blickpunkt des Servers nicht mit dem des Client übereinstimmt muss dieser auf gespeicherte Bilder zurückgreifen. Für dynamische Szenen allerdings müssen immer Referenzbilder erzeugt werden, solange die Bewegungen im Bild nicht zu groß sind. Das kann anhand der Geometrie oder der Bilder bestimmt werden.

Um das Aufdeckungsproblem durch multiple Tiefenbilder auch für mobile Geräte zu beheben wird in [16] ein Framework vorgestellt. Darin wird ein Proxy Server dem System hinzugefügt, der die Bilder des Renderservers verarbeitet, bevor diese an das Zielgerät gesendet werden. Der Proxy Server übernimmt über die parallele Verarbeitung durch die Grafikhardware die Bestimmung der beiden Referenzbilder und das anschließende Warping und Zusammensetzen der Bilder.

3.6 CloudLight

CloudLight [3] ist ein von NVIDIA entwickeltes System zur Erforschung indirekter Lichtberechnung für eine große Anzahl an Clients, um diese in einem verteilten Serversystem bereit zu stellen. Dabei steht vor allem die Umsetzbarkeit und Erweiterung bestehender globaler Beleuchtungssysteme innerhalb eines Cloud-Serversystems im Vordergrund. Es werden drei verschiedene Beleuchtungsverfahren implementiert, welche in einer Cloud unterschiedliche Rendering Pipelines ermöglichen.

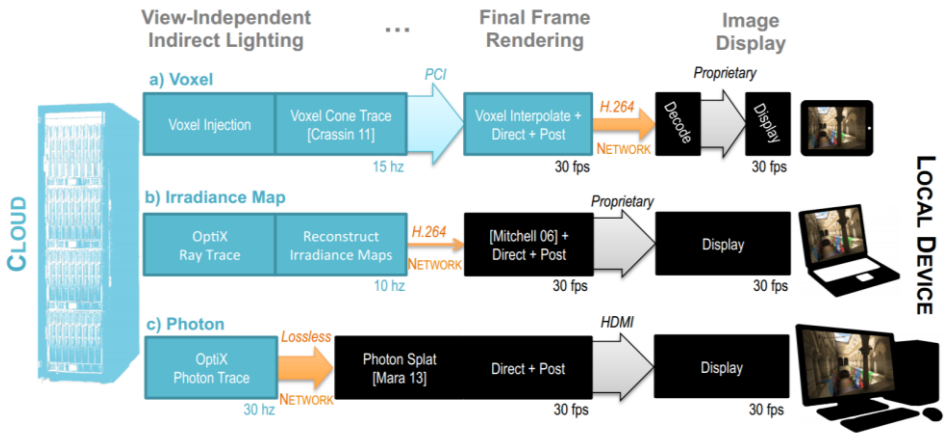


Abbildung 23: Aufbau der drei Pipelines der CloudLight

3.6.1 Irradiance Maps

Der Server erstellt mit Hilfe eines Raytracers die Irradiance Maps für den Client. Ein sehr aufwändiger Berechnungsschritt ist die Parametrisierung der Geometrie, um diese in der Irradiance Map abzubilden. Diese Last kann in dem Fall vom leistungsstarken Serversystem übernommen werden. Der Empfänger muss die eingehenden Karten nur dekodieren und über einen einfachen gather-Algorithmus das Indirekte Licht der direkten Beleuchtung der Szene hinzufügen. Der Rechenaufwand wird hier auf Seiten der Empfänger gering gehalten und belastet die Netzwerkschnittstelle nur wenig. Die statischen Karten lassen sich für verschiedene Empfänger erneut einsetzten, das eine ständige Neuberechnung überflüssig macht. Allerdings stellt die Qualität des Verfahrens nicht das Optimum dar.

Um mehrere Aktualisierungen der Irradiance Maps jede Sekunde durchführen zu können, wird die Berechnungen vereinfacht. Die Karte wird vorab in grobe Basisfunktionen zerlegt, für welche die Beleuchtung mit weniger Strahlen bestimmt werden kann. Die Verwendung von Irradiance Maps, zu meist statischer, Beleuchtung der Geometrie findet heutzutage häufig Anwendung in kommerziell genutzten Engines. Programme, die diese Engines nutzen, lassen sich einfach erweitern, da die Methoden zur Berechnung bereits vorliegen. Eine Aktualisierung der Beleuchtung braucht nur vorgenommen werden, sobald der Server eine neue Karte gesendet hat. Der Berechnungsaufwand bleibt als durchschnittlich anzusehen und das Verfahren liegt von der Belastung des Netzwerks auf vergleichbarem Niveau zu Videostreams. Probleme dieses Verfahrens stellen bisher die komplexe Berechnung der Parametrisierung der Geometrie dar, welche für die Basisfunktionen benötigt werden.

3.6.2 Voxels

Das voxelbasierte Verfahren erzeugt ein komplettes Voxelmodell der Szene. Dieses erlaubt eine umfangreiche parallele Nutzung der Daten von mehreren GPUs zur Berechnung von indirektem Licht für verschiedene Nutzer. Die eigentliche Berechnung des Lichts ist im Kontext einer Voxelhierarchie relativ gering. Die große Datenmenge, die das Voxelgitter aufbringt, lässt für das Verfahren nur das Versenden von vollständig gerenderten Bildern zu.

Um die Architektur der Cloud nutzen zu können, wird das Voxelgitter vorab mit Hilfe eines Cone Tracers blickwinkelunabhängig mit Beleuchtungsinformationen gefüllt. Für die Berechnung der Bilder jedes Nutzers wird das Gitter an jeweilige CPUs weitergesendet. Dafür wird das Voxelgitter komprimiert. Es werden zum Schluss nur Bereiche von hoher Detailmenge mit der vollen Auflösung gerendert. Die vom Client aufzubringende

Grafikleistung beschränkt sich in dem Verfahren auf das Dekodieren und die Darstellung der empfangenen Bilder. Das Niveau der Datenauslastung liegt auf dem eines Image Imposters.

3.6.3 Photons

Für ein photonenbasiertes Cloudrendering Verfahren kommt serverseitig ein aktueller Photonmapper zum Einsatz, basierend auf einem eigenen Raytracing System. Der Server bündelt Photonen anhand des Ursprungs und der Richtung. Das hat den Vorteil der besseren Speicherreservierung und besseren Nutzung der Datenbandbreite des Netzwerkes. Der Client wertet mit den Photonen einen Puffer für die indirekte Beleuchtung über einen Deffered Renderer auf einer niedrigeren Auflösung aus. Im Anschluss muss der Empfänger die alten Photonen auswerten und im Falle von Änderungen der Geometrie neu berechnen. Dadurch, dass nur aktualisierte Geometrie neu berechnet wird, kann weiter Bandbreite und Rechenleistung gespart werden.

Die Größe der gebündelten Photonen wird von der Netzwerkbandbreite bestimmt, sodass die Sendung nicht mehr als eine Frame verzögert wird. Die Ergebnisse zeigen, dass das Verfahren sehr viel Datenverkehr erzeugt, das insbesondere im Kontext von einer größeren Anzahl von Clients Netzwerkkapazitäten überschreitet. Zusätzlich ist im Vergleich zu den anderen Verfahren ebenso der Rechenaufwand auf Seite der Klienten bedeutend größer. Der Berechnungsaufwand entspricht nur dem Drittel eines Photonmappers auf einem lokalen System.

4 Implementation

In diesem Kapitel werden das System, auf dem die eigene Versuchsreihe aufbaut, und die einzelnen Stationen der Implementationen beschrieben. Für jede Station wird das Konzept beschrieben, nach welchen Kriterien oder Ideen die Verfahren ausgewählt wurden. Weitere Beschreibungen über die Umsetzungen und deren Ergebnisse geben weitere Einblicke in die Vorgehensweise und die Vor- und Nachteile der Stationen.

4.1 Testumgebung

Als Setup für die Implementation der verschiedenen Algorithmen wird eine kleine interaktive 3D Szene mittels des OpenGL Frameworks erstellt. Der Aufbau der Szene wird gering gehalten, um die Ergebnisse unabhängig der Rechenleistung des Testcomputers zu betrachten. Es wird eine statische Szene mit ca. 75000 Dreiecken gewählt. Die Interaktivität beschränkt sich auf die freie Navigation des Benutzers, da dies ausreichend für die Untersuchung der Verfahren ist.

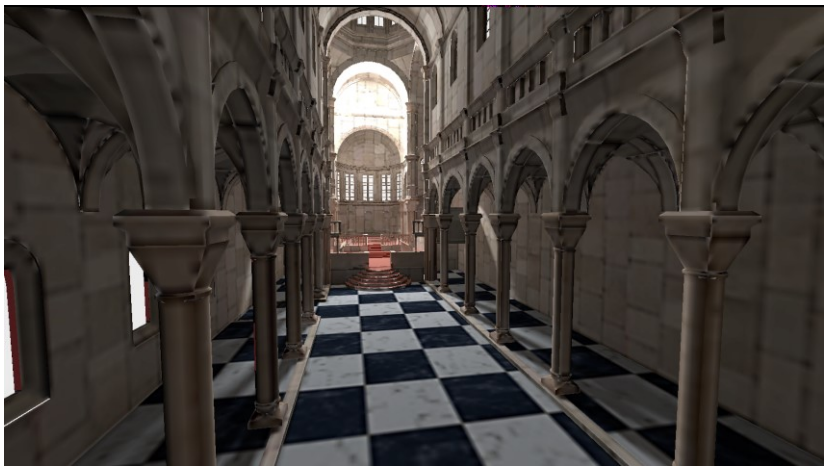


Abbildung 24: Einfaches Rendering mit vorberechneter globaler Beleuchtung

Als Testsystem kommt ein Notebook mit einem Intel® i7-4700MQ mit 2,4GHz getakteten Prozessor, 8 GB DDR3 Arbeitsspeicher und einer NVIDIA Geforce GTX 765M mit 2GB DDR5 Speicher zum Einsatz. Als Softwareversion wird OpenGL/GLSL in der Version 4.3 verwendet.

Die Verzögerung der Client-Server-Kommunikation wird mittels eines Buffers simuliert. Dies ist hinreichend für die Untersuchung der durch die Latenz auftretenden Effekte, deren Ausprägung frei gewählt werden kann. Ein Framebuffer, welcher den Server ersetzt, speichert die gerenderten Bilder und die dazugehörigen Modelview-Matrizen in

einen jeweiligen Puffer. Weitere Framebuffer, die die Berechnungen des Clients simulieren, greifen auf die Texturen und Matrizen an ausgewählten Indizes zu. Abhängig von der gewählten Position für den Zugriff auf den Puffer entsteht die Verzögerung. Bei einer Bildrate von 60 Bildern pro Sekunde ergibt sich bei Zugriff auf den Index 30 des Buffers eine theoretische Verzögerung von 500ms, bei Position 60 1 Sekunde. Durch niedrigere Bildraten bei steigendem Berechnungsaufwand wird die Position des Zugriffs entsprechend gewählt. Bisherige Image Imposter Systeme können Bilder mit einer Verzögerung von 50 bis 150 Millisekunden ausgeben. Da die auftretenden Artefakte mit kleineren Latenzen geringer ausfallen wird für die in den Untersuchungen eine etwas größere Latenz von ungefähr 200 Millisekunden gewählt. Um die Auswirkungen von höheren Latenzen oder Schwankungen in einem Netzwerk zu untersuchen, werden zwei weitere Größen von Verzögerungen von 400 und 600 Millisekunden für die Untersuchungen festgelegt.

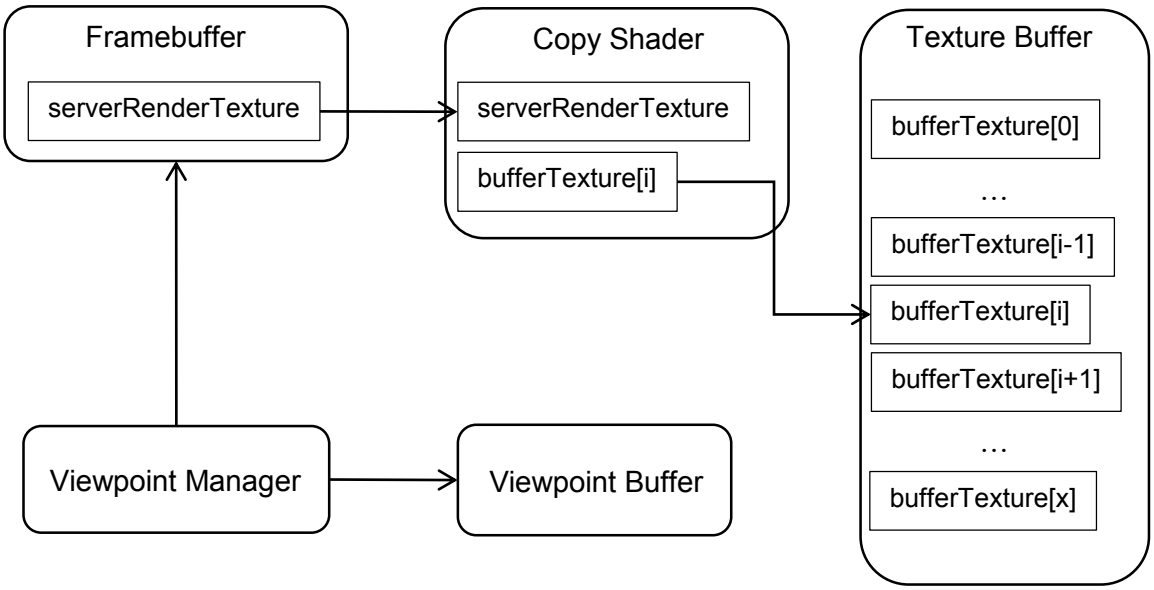


Abbildung 25: Aufbau des simulierten Servers

Ein zusätzlicher Copyshader kopiert die Texturobjekte des Framebuffers in das aktuelle Objekt des Buffers. Eine Zählervariable, die pro Bild inkrementiert wird, bestimmt die aktuelle Position. Bei dem Shader handelt es sich um einen Compute Shader. Auch der Blickpunkt wird in einen separaten Buffer gespeichert. Dieser besitzt die gleiche Länge des Texturbuffers und wird immer an der gleichen Position geschrieben. Der Berechnungsaufwand zur Ausführung des Kopiervorgangs liegt bei ca. 1 Millisekunde (ms).

4.2 Warping

4.2.1 Konzept

Der in Kapitel 3.4 beschriebene Ansatz eines Image Imposters mit der Erweiterung von Tiefenbildern zur Nutzung von 3D Warping bietet einen vielversprechenden Ansatz in einem RRS gegenüber den anderen in dieser Arbeit beschriebenen Verfahren. Image Imposter finden durch ihre geringe Bandbreite schon praktische Anwendungen. Das Hinzufügen der Tiefenwerte erhöht die benötige Bandbreite nur gering und das Warping ermöglicht direkte Umsetzung von Benutzereingaben. In 3.4 werden zwei verschiedene Methoden für das Warping beschrieben. Der direkte Ansatz erzeugt auffällige Artefakte, welche die Bildqualität signifikant verschlechtern. Die zweite Methode erzeugt gegenüber der ersten ein geschlossenes, lückenloses Bild. Demgegenüber entstehen in Bildlücken sogenannte „Rubber Sheets“. Um diese fehlerhaften Bereiche bearbeiten zu können, muss ein zusätzlicher Berechnungsschritt hinzugefügt werden, der diese Bildbereiche erkennt und gegebenenfalls für die weitere Manipulation extrahiert oder markiert. Um diesen erhöhten Berechnungsaufwand zu vermeiden sollen in dieser Arbeit Ansätze getestet werden, welche die Artefakte der direkten Methode adressieren. Das erlaubt einen kompletten Überblick über die entstehenden Fehler und vereinfacht die Implementierung von Lösungen.

4.2.2 Umsetzung

Um die Geschwindigkeit der Grafikkarte auszunutzen und keinen Nachteil gegenüber des Meshbasierten Ansatzes zu bekommen wird das Warping in einem Shader umgesetzt. Das Warping verlangt das Schreiben von beliebigen Koordinaten einer Textur.

Pseudocode

input: mvp of current frame, mvp of reference frame, reference frame, depth

output: derived frame

```
1. forall pixel do
2.         calculate pixel from screenspace to normalized device coordinate;
3.         oldpoint = current pixel.xy, depth, 1.0;
4.         remove projection of oldpoint, add new projection and homogenize
5.         calculate newpoint to screenspace;
6. return newpoint.xy;
```

Algorithmus 1: Warping Shader

Dieses Kriterium erfüllen Compute Shader. Diese, für GPGPU entwickelten Shader erlauben den parallelisierten Zugriff auf Indices verschiedener Pufferobjekte. In diesem Fall werden die vom vermeintlichen Server erzeugten Texturen ausgelesen.

Der Shader bekommt als Input das vom Server gerenderte Bild, die dazugehörige Model-View-Projection-Matrix und die aktuelle Model-View-Projection-Matrix. Die erstellten Punkte *oldpoint* in Algorithmus 1 entsprechen 3D-Euklidischen Punkten. Aus diesem Grund handelt es sich bei der verwendeten Tiefe *depth* um den homogenisierten Tiefenwert des Tiefenpuffers. Das eingehende Bild besteht als RGBA Textur. Zur Optimierung sind die Tiefenwerte in den Alpha Kanal der Textur eingetragen, da dieser nicht genutzt wird. Bevor die neu projizierten Punkte *newpoint* wieder zurück in den Bildbereich transformiert werden können, müssen diese wieder in euklidische Punkte umgewandelt werden. Der Zugriff auf die einzelnen Texel der eingehenden Referenztextur wird über die gl_WorkGroupID und die gl_LocalInvocationID der benutzen Threads gesteuert. Es werden die Farbwerte in die Zieltextur geschrieben, die in der Referenztextur an der Xy-Koordinate des neuberechneten Punktes gelesen werden.

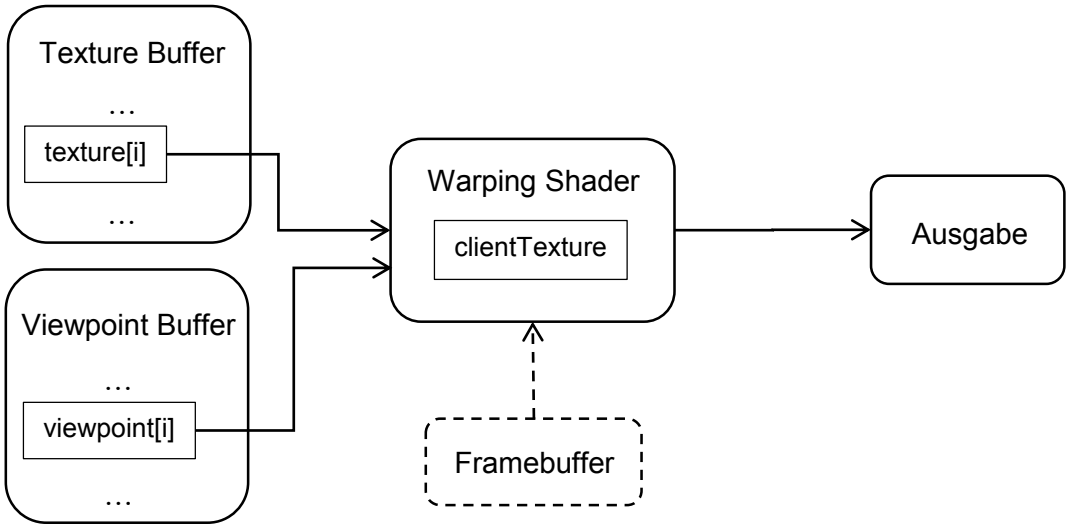


Abbildung 26: Aufbau des simulierten Clients

Ein zusätzlicher Framebuffer kann dem Client hinzugefügt werden um dem Warping mehr Daten zur Verfügung zu stellen. Dies wird ausführlicher in der Bewertung beschrieben. Die Ausgabe besteht ebenfalls aus einem Framebuffer, der die Texturen des Warping Shaders in ein bildschirmfüllendes Dreieck rendert.

4.2.3 Bewertung

Wie schon in 3.4 genannt treten qualitative Mängel ein. Lösungsmöglichkeiten dazu sollen in den folgenden Unterkapiteln beschrieben werden.

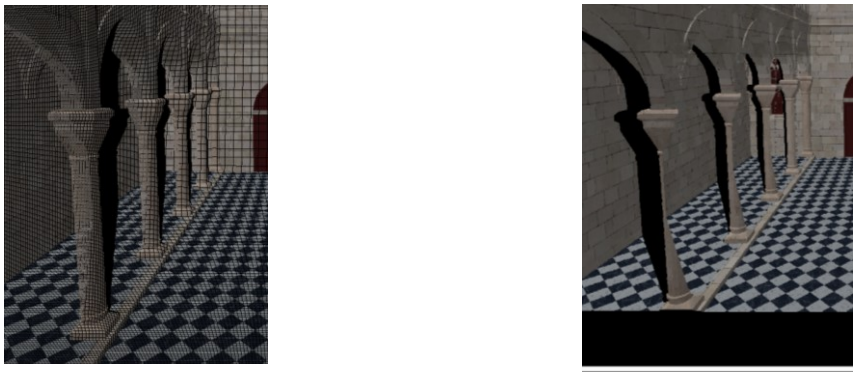


Abbildung 27: Warping mit Compute Shader ohne Tiefentest

Da Compute Shader keine Möglichkeit bieten einen Pixel mehrfach zu lesen und zu schreiben, muss der Tiefentest im Falle der mehrfachen Adressierung eines Pixels in einen weiteren Schritt nachträglich ausgeführt werden. Hier werden die Bildpunkte erneut gewarpt und der zu schreibende Pixel mit dem vorhandenen Pixel an der entsprechenden Bildkoordinate verglichen und gegebenenfalls ersetzt. Eine alternative Methode löst dieses Problem durch die Bestimmung der Reihenfolge der Warpingschritte. Hierbei werden vorab die Epipole des abgeleiteten Bildes im Referenzbild bestimmt. Über die Eigenschaften des Punktes wird festgelegt, in welche Bereiche das Referenzbild unterteilt und in welcher Reihenfolge die Punkte abgetastet werden [17].

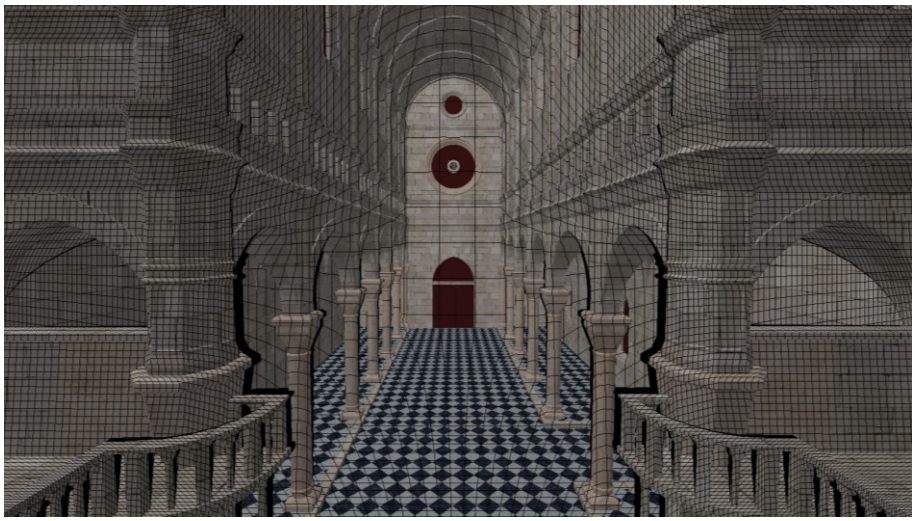


Abbildung 28: Vorwärtsbewegung der Kamera bei ~200ms Latenz

Abbildung 29: Vorwärtsbewegung der Kamera bei ~400ms Latenz

Abbildung 30: Vorwärtsbewegung der Kamera bei ~600ms Latenz

Basierend auf der Annahme, dass es sich bei dem Client um ein System mit ausreichend Rechenleistung handelt, lässt sich die Tiefe ebenso auf dem Client berechnen. Der Client muss einen Renderpass mit der vollständigen Geometrie durchführen. Dies ist für mittelstarke bis starke Hardware eine mögliche Option. Generell hängt das von der Anzahl der Grafikprimitive ab. Die Berechnung der Tiefe auf dem Testgerät beträgt ca. 3ms.

Abbildung 28 – 30 zeigen die auftretenden Artefakte des direkten Warpings und deren Ausprägung bei zunehmender Latenz. In allen drei Bildern ist der Tiefentest integriert. Die durch die fehlende Interpolation entstehenden Lücken nehmen mit steigender Latenz zu und lassen das Bild dunkler wirken. Auch fällt es schwieriger Objekte zu erkennen. Die Flächen der leeren Bereiche hinter Objekten nehmen ebenso mit der Latenz zu, welches insbesondere bei Seitwärtsbewegungen (Abbildung 25) auffällt.

Neben den Silhouetten von Vordergrund Objekten im Hintergrund entstehen weitere große Flächen ohne Informationen. Diese treten bei Seitwärtsbewegungen der Kamera

am Rand auf, bei Rückwärtsbewegungen an allen Bildrändern oder bei der Rotation der Kamera (Abbildung 26).

Der Aufwand des Algorithmus liegt bei O(n) und ist alleine durch die Auflösung des eingehenden Bildes bestimmt. Auf Grund dessen ist diese Technik für alle drei Kategorien aus 1.3 interessant, allerdings erfordert es weitere Berechnung, um die qualitativen Mängel zu beheben, weshalb die erste Kategorie für das Verfahren auszuschließen ist. Eine Messung ergibt eine durchschnittliche Berechnungszeit von 2ms.

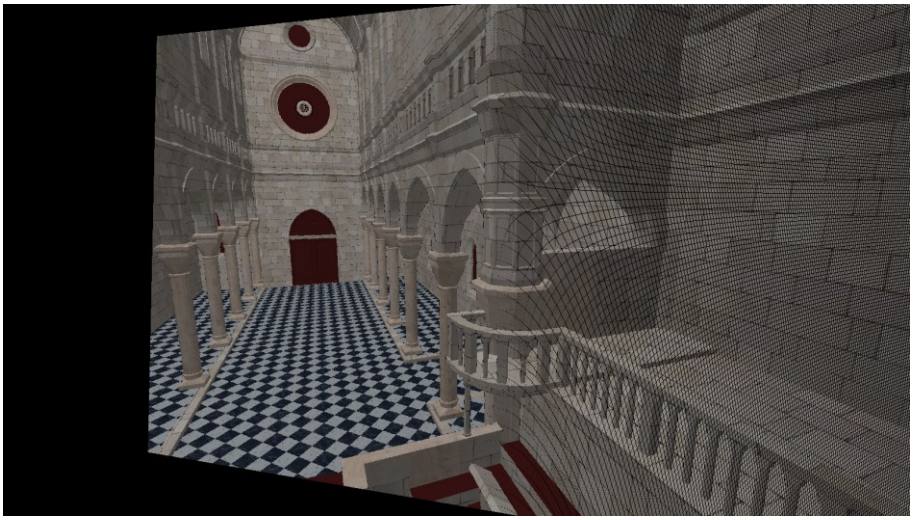


Abbildung 31: Rotation der Kamera bei ~400ms

Abbildung 32: Seitwärtsbewegung der Kamera bei ~400ms

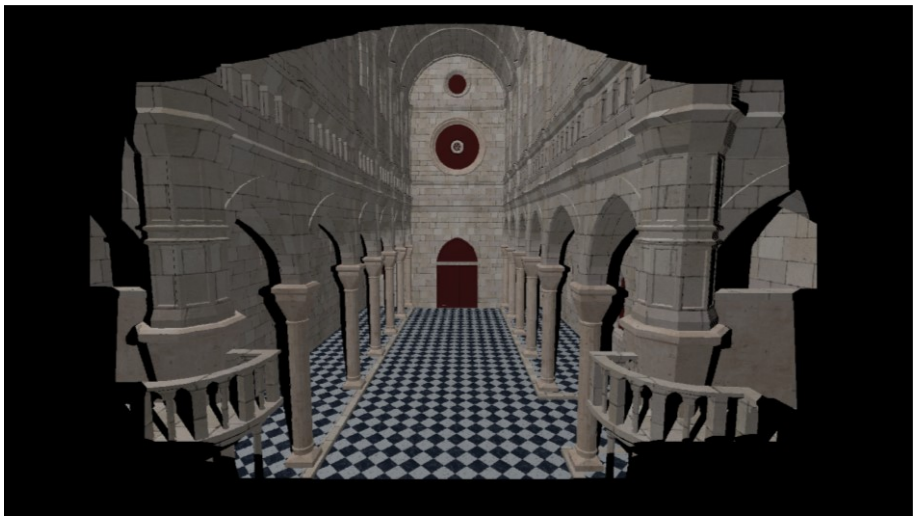


Abbildung 33: Rückwärtsbewegung der Kamera

4.3 **Abtastungsproblem**

4.3.1 Konzept

Da die vorhandene Bildinformation limitiert ist kommt es, wie im vorherigen Unterkapitel gezeigt, zu lückenhaften Fehlern. Diese Effekte entstehen für Oberflächen, welche durch die veränderte Modelview-Transformation nach dem Warping näher an der Kamera liegen und somit vergrößert werden. Beispielsweise wird eine Fläche durch eine Vorwärtsbewegung der Kamera doppelt so groß abgebildet. Für die Fläche stehen nur die Werte aus der originalen Größe, also der Hälfte der Fläche zur Verfügung. Die Oberflächenpunkte werden durch den Warpingalogrithmus gleichmäßig in der neuen Fläche verteilt. Eine einfache Möglichkeit die Lücken zwischen den abgebildeten Punkte zu füllen, ist es diese mit den interpolierten Werten der Nachbarpunkte zu füllen.

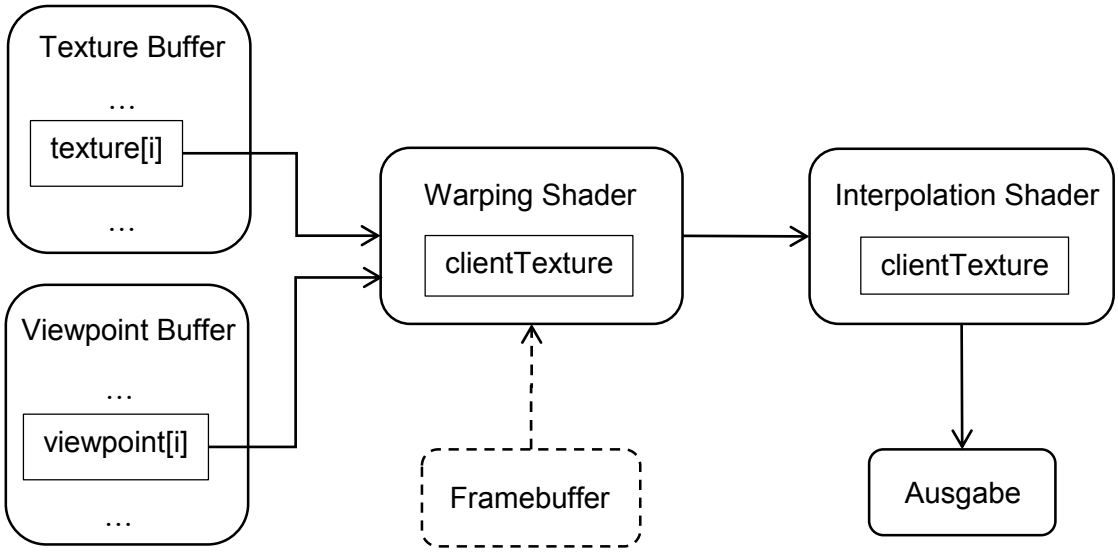


Abbildung 34: Aufbau der Client Pipeline mit Interpolation

Bei dem Interpolation Shader handelt es sich ebenso um einen Compute Shader. Dieser faltet das Bild für jeden Pixel mit einer Matrix und mittelt die gesammelten Werte. Zur Optimierung kann die Ausgabetextur des Warping Shaders für die Interpolation überschrieben werden. Auch werden nur die Pixel überschrieben, in welcher kein Farbwert eingetragen ist. Innerhalb des Interpolation Shaders lässt sich, wie im vorhergehenden Kapitel beschrieben, der Tiefentest integrieren.

Wird in einem Remote Rendering Setup ein hybrides Verfahren verwendet ergibt sich eine weitere Möglichkeit auftretende Lücken zu reduzieren. Anstatt des Warpingverfahrens, das die Koordinaten des Referenzbildes in die Zieltextur abbildet, werden die Koordinaten in der Zieltextur zurückgerechnet. Das Verfahren erfordert keinen zusätzlichen Shaderaufruf und erhöht die Präzision der Abtastung, da für jeden Punkt einer Oberfläche ein korrespondierender Punkt gesucht wird.

4.3.2 Umsetzung

Durch die Gleichmäßigkeit und die limitierte Bewegungsgeschwindigkeit der Kamera entstehen ausschließlich Lücken von einem Pixel Breite. Eine Faltungsmatrix mit einem 3x3 großen Kern filtert für jeden nicht geschriebenen Bildpunkt in 8ter-Nachbarschaft den Medianwert und schreibt diesen in die Stelle. Um das Überschreiben von schwarzen Objekten und Texturen zu vermeiden, wird beim Löschen der Zieltextur im Alpha Kanal ein bestimmter Wert geschrieben. Auf diesen Wert kann vor jeder Filterung getestet werden. Eine untere Grenze für gefilterte Bildpunkte soll Interpolationsartefakte an Rändern zu leeren Bereichen verhindern.

Für die Methode der Rückberechnung der Referenzkoordinaten wird der Warping Shader angepasst.

Pseudocode

input: mvp of current frame, mvp of reference frame, reference frame, current depth

output: derived frame

1. **forall** pixel **do**
2. calculate pixel from screenspace to normalized device coordinate;
3. newpoint = current pixel.xy, current depth, 1.0;
4. remove projection of newpoint, add old projection and homogenize
5. calculate oldpoint to screenspace;
6. **return** oldpoint.xy;

Algorithmus 2: Rückberechnung Warping

In dem Shader wird die Zieltextur traversiert. Für die verwendete Tiefe wird die Tiefenkarte des Framebuffers des Clients genutzt. Für die Deprojizierung des Punktes durch die Model-View-Projection-Matrix des Referenzbildes wird die aktuelle MVP des Clients verwendet. Für die erneute Projektion wird die MVP des Referenzbildes genutzt. Mit den neu berechneten Koordinaten werden die Farbwerte im Referenzbild ausgelesen und in die Zieltextur geschrieben. Ein Tiefentest zwischen dem gefundenen Punkt und der berechneten Tiefe verhindert das Schreiben in aufgedeckte Bereiche.

4.3.3 Bewertung

Im Rahmen der Untersuchung kann die Filterung das subjektive Erscheinungsbild signifikant steigern und die meisten der Interpolationsfehler ausgleichen. Bei schnellen Kamerabewegungen hingegen kann das Verfahren nicht alle Lücken abdecken und es treten weiterhin Interpolationsfehler auf. Das ist in Abbildung 33 deutlich zu erkennen. Eine Vergrößerung der Filtermatrix verringert das Aufkommen der Fehler nur minimal, da in den Problembereichen die Datenmenge zu gering ist.

Die Methode des hybriden Systems zeigt bei allen getesteten Latenzen keine Lücken in Oberflächen. Einzig die Aufdeckungsproblematik bleibt bestehen. Das Verfahren macht die Versendung der Tiefenwerte des Referenzbildes zum Client nicht erforderlich. Dennoch ist dies sinnvoll, um das unbeabsichtigte Schreiben von Farbwerten in die aufgedeckten Bereiche zu vermeiden.

Der Aufwand der Methode ist abhängig von der Kamera und den dadurch entstandenen Interpolationslücken. Die Berechnungen für jede gefundene Lücke sind gleich. Folglich ist der Aufwand vergleichbar zu dem des Warping. Die gemessene Berechnungszeit liegt bei ca. 3ms.

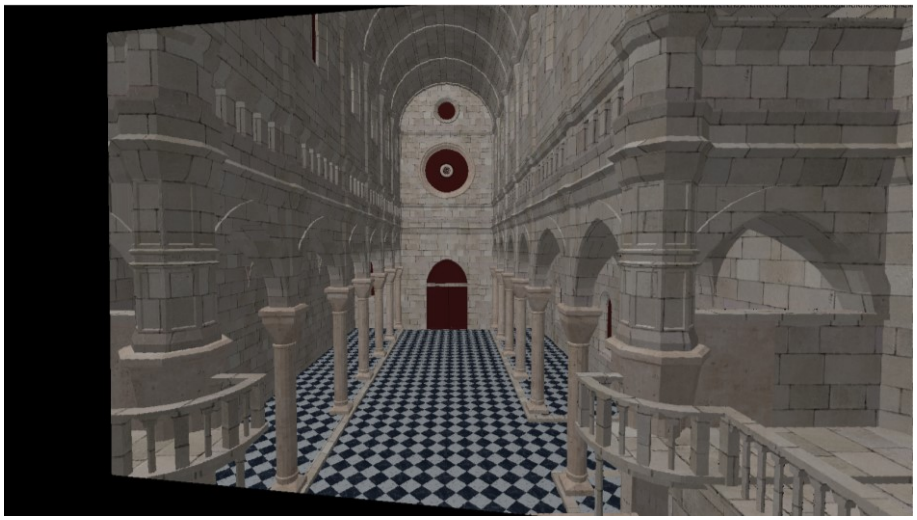


Abbildung 35: Rotation der Kamera mit Interpolation

Abbildung 36: Vorwärtsbewegung mit Interpolation bei ca. 200ms Latenz

Abbildung 37: Vorwärtsbewegung mit Interpolation bei ca. 400ms Latenz

Abbildung 38: Vorwärtsbewegung mit Interpolation bei ca. 600ms Latenz

Abbildung 39: Warping mit Rückberechnung

4.4 Randbehandlung

4.4.1 Konzept

Die durch Rotationen, Rück- oder Seitwärtsbewegungen der Kamera hervorgerufenen Artefakte im Randbereich lassen sich separat betrachten. Da sich die dort benötigten Bildpunkte außerhalb des gesendeten Bildes befinden, fehlt diese Punkte vollständig. Eine Möglichkeit ist es, diese Information zusätzlich zu senden, indem das Referenzbild um einen bestimmten Faktor zum Ausgabebild vergrößert wird. Dabei bleibt zu untersuchen, wie groß der Faktor sein muss, um das Artefakt für maximale Latenzen zu minimieren.

4.4.2 Umsetzung

In dem Verfahren wird für alle Framebufferobjects und Compute Shader, die auf dem kompletten Bildbereich arbeiten, die Auflösung schrittweise erhöht. Allein der Viewport des Renderingpass der bildschirmfüllenden Ausgabe wird auf die kleine, gewünschte Zielauflösung gesetzt und innerhalb der großen Auflösung zentriert.

4.4.3 Bewertung

Mit einem Anstieg der Latenz muss die Auflösung entsprechend wachsen, um die leeren Bereiche am Bildrand füllen zu können. In Anbetracht dessen würde natürlich der Datendruchsatz im Netzwerk entsprechend gesteigert und entsprechend auch die Latenz. Dies steht entgegen den in 1.2 gesetzten Zielen. Zusätzlich wird der Berechnungsaufwand für das Abbilden gesteigert. Generell lässt sich die Verwendung der Methode nicht ausschließen. Die Auflösung muss am verwendeten System und Netzwerk bestimmt werden. Sie erscheint für niedrige Latenzen als praktikable Möglichkeit leere Randbereiche zu minimieren.

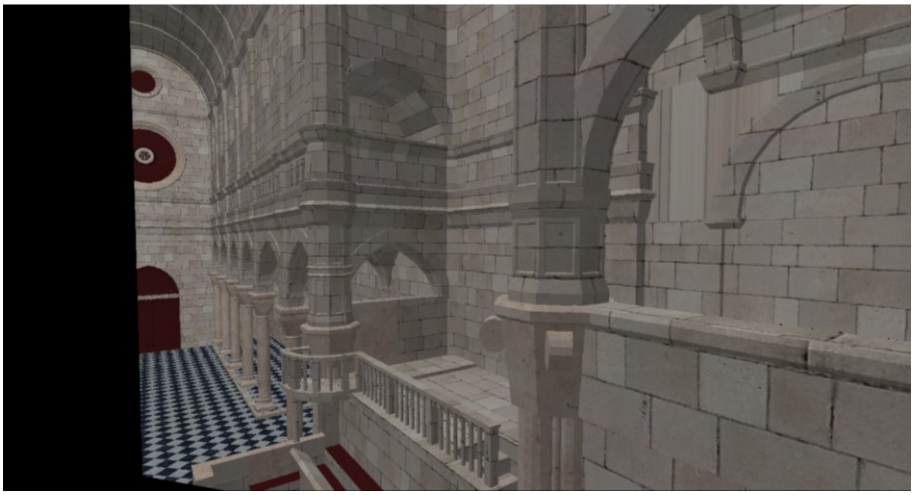


Abbildung 40: Rotation bei ca. 200ms Latenz

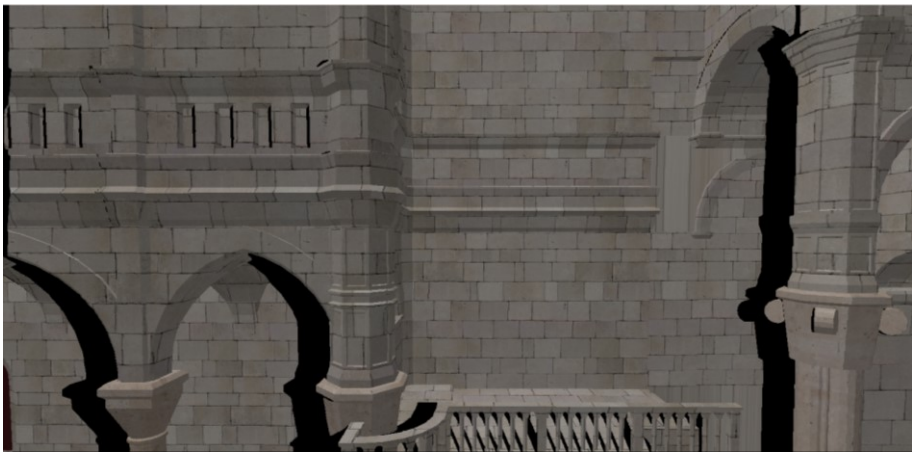


Abbildung 41: Seitwärtsbewegung bei ca. 200ms Latenz

(a)

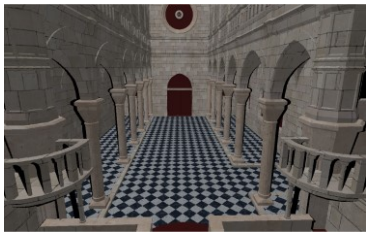
(b)

(c)

(d)

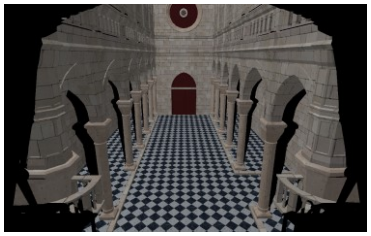
(e)

(f)



(g)

(h)



(i)

Die Größe der Referenzbildern Abbildungen (a), (b) und (c) liegt bei 1280 x 800 und ist gleich der Ausgabegröße. Die Ausgabegröße ist in allen 9 Testfällen gleich. Für die Bilder (d), (e) und (f) wurden die Referenzbilder auf 1600 x 900 vergrößert. In den Bildern (g), (h) und (i) liegt die Größe der Referenzbilder bei 1920 x 1080. Die Latenz liegt in den Bilder (a), (d) und (g) bei 200ms. In den Bilder (b), (e) und (h) liegt die Verzögerung bei 400ms und für die Bilder (c), (f) und (i) bei 600ms.

Abbildung 42: Seitwärtsbewegung bei ca. 400ms Latenz

Für eine Latenz von 200ms ist eine 1,5 fache Auflösung ausreichend, um Randbereiche bei Kamerabewegungen zu füllen. Für eine Latenz von 400ms muss die Auflösung bereits verdoppelt werden.

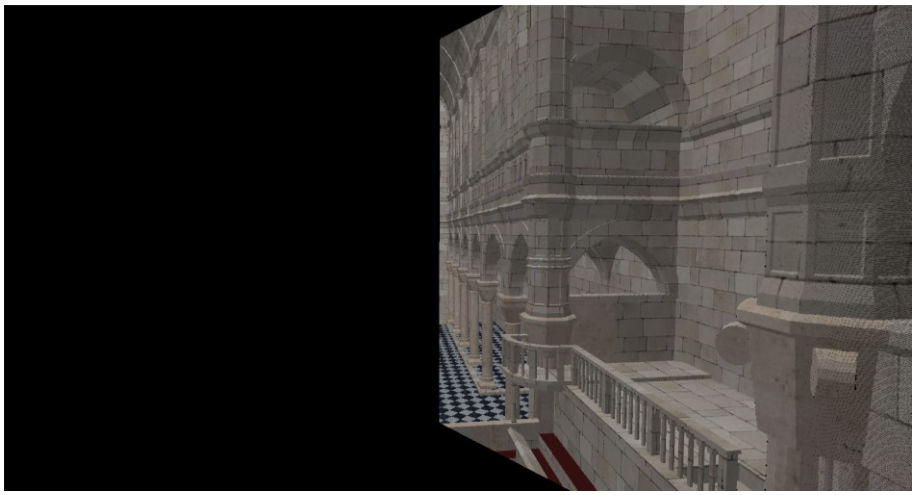


Abbildung 43: Rotation bei ca. 400ms Latenz

4.5 Holefilling

4.5.1 Konzept

Die durch das Aufdeckungsproblem entstehenden Artefakte sind die auffälligsten Qualitätsminderungen des Warpings. Dabei bleiben die Silhouetten der vordergründigen Objekte als leerer Bereich sichtbar.

Innerhalb der leeren Bereiche können Objekte liegen, die bisher nicht sichtbar waren. Diese Information liegt beim Client nicht vor. Eine Option ist es, ein vergleichbares Model auf dem Client zu berechnen und dieses so weit wie möglich im Berechnungsausfand zu reduzieren. Die Reduktion kann von der Leistungsfähigkeit des Clients abhängig gemacht werden. Dabei stehen beispielsweise Polygonanzahl, Shading oder Auflösung zur Disposition. Ein interessanter Anwendungsfall für das Remoterendering ist die Integration eines qualitativ schlechteren Bildes des Client in das abgebildete, qualitativ hochwertige Bild des Servers. In der Anwendung wird die Auswirkung von reduzierter Geometrie und Texturauflösung untersucht.

4.5.2 Umsetzung

Das reduzierte Modell wurde mit Blender[11] aus dem originalen Modell erstellt. Die Anzahl der Polygone wurde auf 20 Prozent verringert. Ein zusätzlicher Framebuffer rendert das reduzierte Modell mit der Modelviewmatrix des Clients in eine eigene Textur. Ein weiterer Compute Shader fügt die Textur in den Bildkkordinaten der leeren Bereiche in das abgeleitete Bild ein. Wie schon in 4.3.2 werden die leeren Bereiche über den im Alphakanal bestimmt.

4.5.3 Bewertung

Ein markanter Vorteil ist ein komplettes Bild, da alle Bereiche gefüllt sind. Negativ hingegen fallen vor allem die niedriger tesselierten Bereiche von runden Objekten aus, die abhängig von der Latenz erst verzögert mit dem detaillierteren Bild des Servers überlagert werden. Die Qualität des Hintergrundbildes kann entsprechend der Kapazität des Clients angepasst werden und erscheint als Kompromisslösung für Kategorie 2 und 3 Zielgeräte.

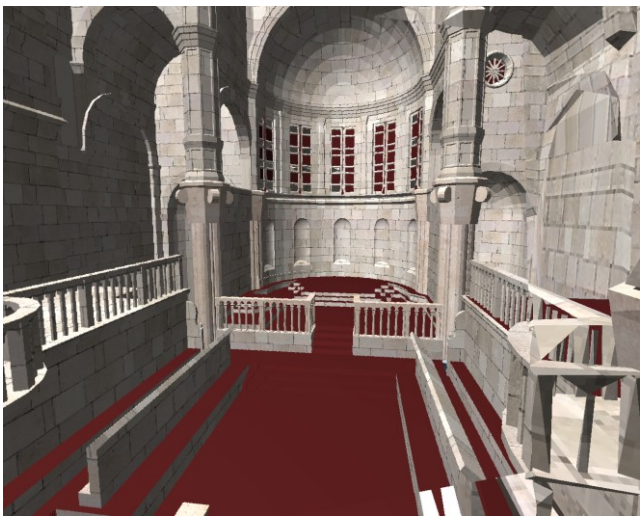


Abbildung 44: Rotation der Kamera und niedrig tesselierter Geometrie im Hintergrund

[11] www.blender.com

Abbildung 45: Seitwärtsbewegung und niedrig tesselierter Geometrie im Hintergrund

4.6 Compositing

4.6.1 Konzept

In diesem Teil der Arbeit wird untersucht, ob sich die Beleuchtungsinformation aus benachbarten Bildpunkten extrahieren lässt, um damit eine plausible Beleuchtungsinformation für Hintergrundpixel abzuschätzen. Um nutzbare Informationen der Szene zu gewinnen, wird auf dem Client die komplette Szene, allerdings ohne jegliche Beleuchtung, gerendert. Das dabei entstandene Bild soll dazu genutzt werden die geometrischen Eigenschaften des zu berechnenden Hintergrundpixels zu vergleichen. Da das Rendern reiner Geometrie und deren Texturen durch die Grafikhardware in der heuten Zeit stark optimiert ist, kann dieser Schritt als Möglichkeit für starke Hardware des Clients herangezogen werden. Als weitere Information soll für jeden zu berechnenden Hintergrundpixel der Abbildungsvektor bestimmt werden. Dieser Wert soll genutzt werden, um die Größe des leeren Bereiches abzuschätzen. Dadurch soll unnötiges Suchen von Lichtinformation in diesem Bildbereich vermieden werden. Das Verfahren ist unabhängig von der Globalen Beleuchtung; der Server sendet einzig das fertig gerenderte Bild, welches vom Client wiederum normal gewarpt wird.

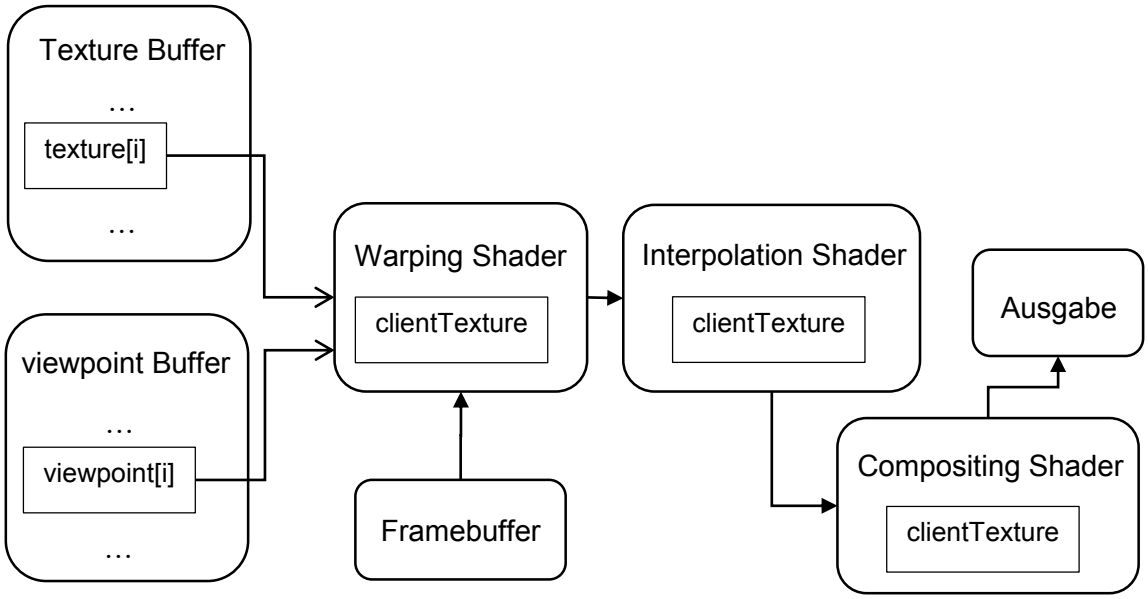


Abbildung 46: Client Pipeline mit Compositing

4.6.2 Umsetzung

Ausschlaggebend für die Beleuchtung eines Oberflächenelements ist dessen Position im Raum und Normale. Für den Bildbereich lässt sich die Position über die Nähe des zu vergleichenden Bildelements und den dazugehörigen Tiefenwert ableiten. Die Normale kann aus dem vom Client gerenderten unbeleuchteten Bild entnommen werden. Das Compositing Verfahren soll möglichst viele Beleuchtungsinformationen von Bildpunkten aus der Umgebung finden. Die Eigenschaften der Punkte dürfen einem zu bestimmenden Differenzwert zu dem Hintergrundpixel nicht überschreiten. Leere Bereiche entstehen, wenn sich Bildelemente eines Objektes mit geringerer Tiefe vor einem anderen Objekt gewarpt werden. Folglich ist das Suchen von Vergleichswerten in Richtung des Abbildungsvektors uninteressant, da sich dort die Bildwerte des voranliegenden Objektes befinden. Aus diesem Grund soll ein Zufallsverfahren entlang des negativen Abbildungsvektors nach möglichen Vergleichswerten suchen. Eine Zufallsverteilung erscheint sinnvoll, da keine Aussage über die Eigenschaften der umliegenden Geometrie gemacht werden kann.

Das Verfahren ist vergleichbar zu dem in 4.5 aufgebaut. Anstelle des reduzierten Modells wird die komplette Szene gerendert. Der anschließende Shaderpass über die Ausgabetextur warpt die Bildpunkte und berechnet den inversen Abbildungsvektor. Ein Zufallsalgorithmus ermittelt eine festgelegte Anzahl neuer Bildkoordinaten in einem festgelegten Abstand. Alle Punkte werden auf ihre geometrischen Eigenschaften vergli-

chen (Tiefe und Normale). Sobald diese in einem Toleranzbereich liegen werden die Beleuchtungsstärken ermittelt. Die gemittelte Summe dieser Werte ergibt die geschätzte Beleuchtung an der gesuchten Stelle.

Pseudocode

input: mvp of current frame, mvp of reference frame, reference frame, depth, background

output: derived frame

1. **Warp positon**
2. **forall** neighbour pixel x **do**
3. **forall** neighbour pixel y **do**
4. Get random position;
5. accumulate lighting at random position;
6. **return** median lighting color;

Algorithmus 3: Compositing

4.6.3 Bewertung

Die Ergebnisbilder zeigen, dass das Verfahren die Diskontinuitäten der Warpingartefakte subjektiv gut reduzieren kann. In diesem Versuch entstehen auffällige Schlieren innerhalb der leeren Bereiche, die dem mangelhaften Zufallsalgorithmus geschuldet ist. Auch schafft es das Verfahren nicht, die Texturfarbe und Beleuchtung vollständig zu trennen. Es lassen sich die Oberflächen der abgetasteten Geometrie in den gefüllten Bereichen wiedererkennen. Sie verändern sich mit der Länge des Warpingvektors. Zusätzlich werden viele Vergleichspunkte benötigt, um einen guten Mittelwert zu finden. Bei

Abbildung 47: Compositing bei ca. 400ms Latenz

Kamerabewegungen, die viele leere Bereiche entstehen lassen, kann das die Rechenleistung des Clients in Anspruch nehmen. Zusätzlich gibt es keine Garantie, dass Vergleichswerte gefunden werden. In diesem Fall werden die leeren Bereiche weiterhin schwarz dargestellt. Festzuhalten ist, dass das Verfahren für niedrige Latenzen die Bildqualität subjektiv stark verbessern, für hohe Latenzen allerdings starken Berechnungsaufwand und nicht alle leeren Bereiche mit Informationen füllen kann. Somit erscheint es für Clients der dritten Kategorie geeignet. Die durchschnittliche Berechnungszeit liegt bei ca. 2ms.

Abbildung 48: Compositing Artefakte

Abbildung 49: Compositing bei ca. 400ms Latenz

4.7 Reflektionen

4.7.1 Konzept

Spiegelungen sind in der Computergrafik Effekte, die abhängig vom Standpunkt des Betrachters sind. Diese müssen innerhalb des Warpingprozesses an die neue Position der Kamera angepasst werden. Für den Fall, dass gespiegelte Objekte bei einer Kamerabewegung nicht angepasst werden, verzerren sich diese entsprechend ihrer bisherigen Position.

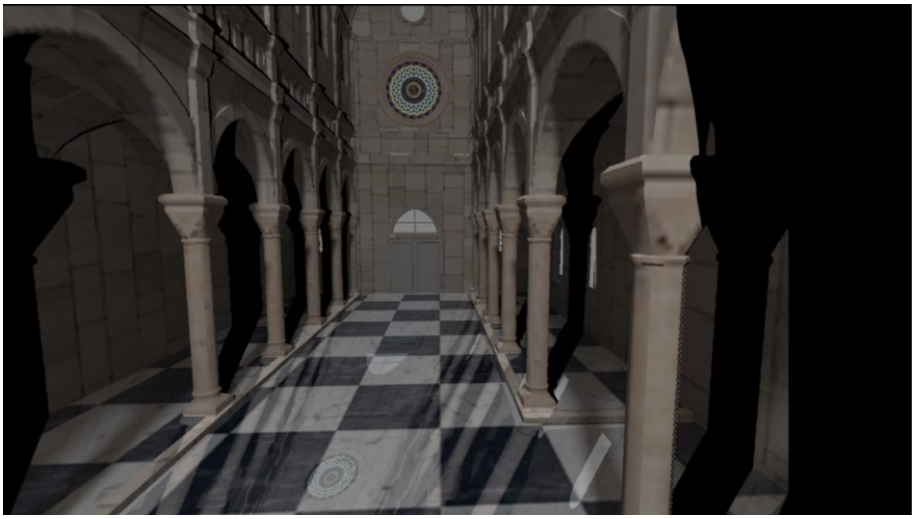


Abbildung 50: Reflektionen ohne Warping

Da verzerrte Spiegelungen einen negativen Einfluss auf das Ergebnisbild haben, müssen diese ebenso gewarpt werden. In Kapitel 2.5 wird ein Verfahren zur Simulation von Reflektionen beschrieben, in dem die Geometrie innerhalb des reflektierenden Objektes in einen Stencil invertiert gerendert wird. Aus diesem Grund lassen sich die Reflektionen ebenso als Geometrie ansehen, die im Bildbereich über Koordinaten und eine entsprechende Tiefe verfügen. Es gilt diese Tiefe zu bestimmen. Die Punkte sollen am Ende ebenso gewarpt werden wie in 4.2. Grundlage soll eine separate Textur sein, welche die Informationen der Reflektionen enthält. Diese wird vom Server erzeugt und an den Client gesendet. Der Client wird nach dem Warping der normalen Szene einen zusätzlichen Shaderpass für das Warping der Reflektionen durchführen, die beiden Bilder kombinieren und anzeigen.

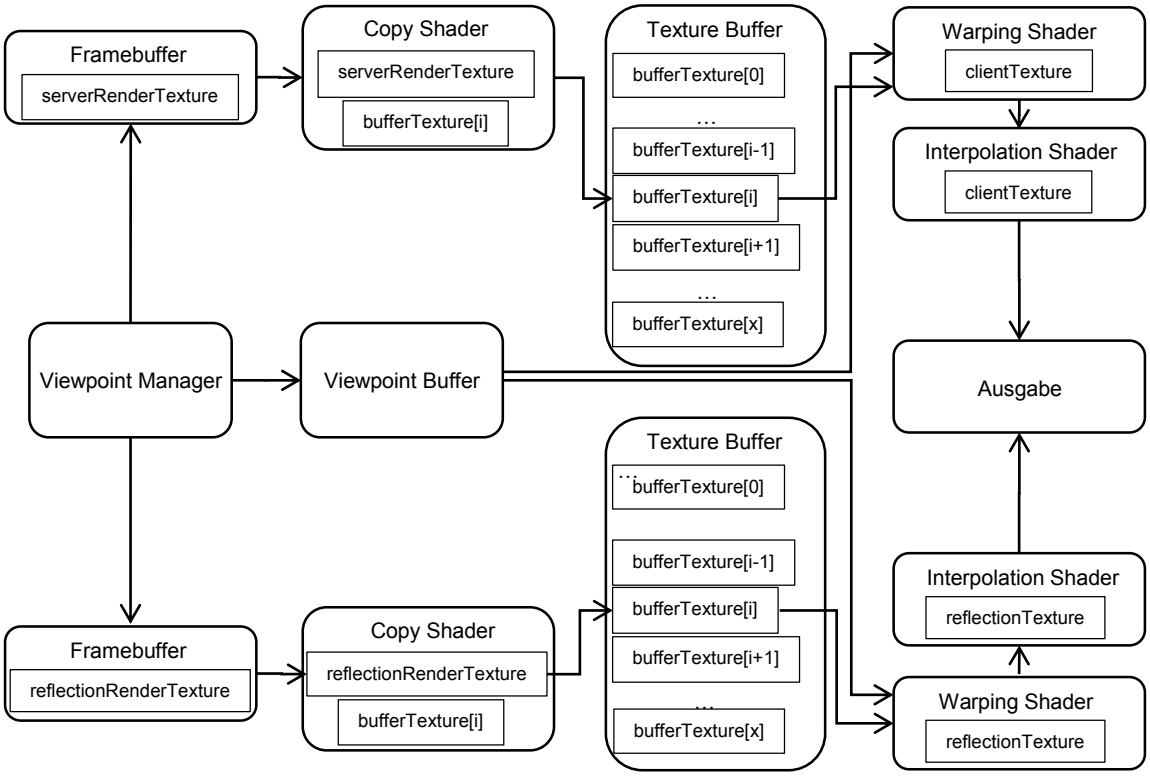


Abbildung 51: Client Pipeline mit Waring von Reflektionen

4.7.2 Umsetzung

Das Verfahren soll unabhängig von der Technik der Reflektionsbestimmung sein. Wegen der simulierten RRS Umgebung und der Berechnungen von Server und Client auf einem Gerät wurde ein schnelles Verfahren gesucht. Im Rahmen dieser Arbeit wird eine Screenspace Reflections (SSR) Verfahren angewendet. Dieses wird serverseitig im Anschluss des normalen Renderingdurchgangs in einem eigenen Framebufferobject durchgeführt und das Ergebnis in eine eigene Zieltextur geschrieben. Die Textur beinhaltet an den entsprechenden Koordinaten die Farbinformation der Reflektion und die Länge des Reflektionsvektors.

Im Bereich des Clients wird die Textur des Framebufferobjectes in einen Compute Shader geladen. Innerhalb dieses Shaders wird ein Warping analog zu der in 2.4 vorgestellten Methode durchgeführt.

In einem letzten Schritt zur Ausgabe des Bildes werden das normal gerenderte Bild und die Textur mit den gewarpten Reflektionen innerhalb eines Fullscreen-Quads aufeinander gerechnet.

4.7.3 Bewertung

Die SSR Methode sucht nur nach Reflektionspunkten im Bildbereich. Deswegen fehlen der Reflektionstextur all jene Informationen von Reflektionen außerhalb des gerenderten Bildes und sie ist folglich nicht vollständig. Das Verfahren der SSR findet mittlerweile in vielerlei Programmen Anwendung; es wäre damit auch auf schnelleren Clients möglich und das Potential eines Servers wäre nicht ausgenutzt. Innerhalb eines RRS ist eine Methode von Interesse, die vollständige Reflektionen liefert. Ein Raytracing Verfahren wäre hierbei interessant. Es benötigt allerdings eine hierarchische Struktur der Szene zur Schnittpunktberechnung, welche vom Server übernommen wird. Für eine Evaluation des Warpings von Reflektionen ist eine, mit einem SSR-Verfahren erstellte, Reflektionstextur vollkommen ausreichend, da einzelne Punkte gewarpt werden und deren Anzahl irrelevant ist.

Abbildung 52: Reflektionen mit Warping

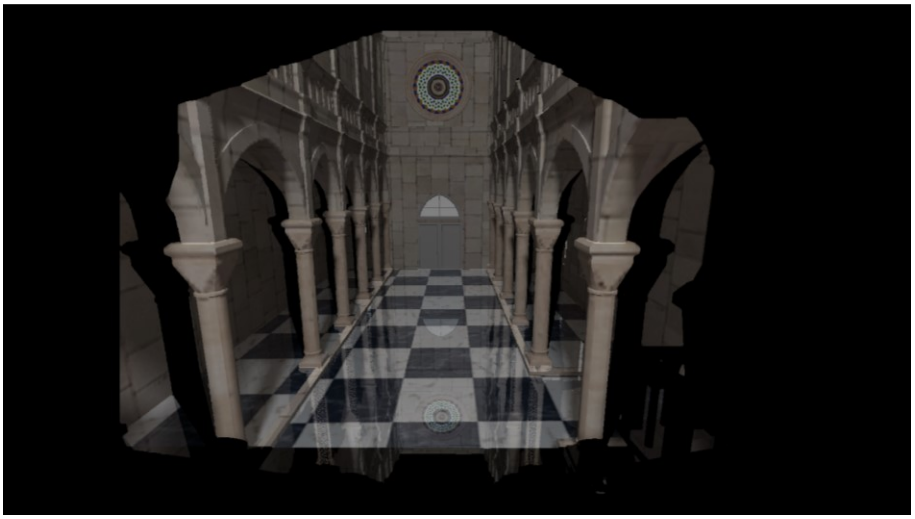


Abbildung 53: Reflektionen mit Warping

Das Warping von Reflektionen ist ausschließlich für planare Flächen geeignet. Dies lässt aber zur Simulation von Glossy Reflections die anschließende Bearbeitung, beispielsweise mit einem Normalmapping Verfahren, zu. In dieser Arbeit wird dieses im finalen Renderdurchlauf auf die Reflektionstextur angewand.

Das Verfahren liegt in der gleichen Aufwandsklasse wie das 3D-Warping. Da die Reflektionen nur in einem Teil des Referenzbildes auftreten, liegt die Berechnung entsprechend darunter. Es wird eine durchschnittliche Berechnungszeit von 1ms gemessen.

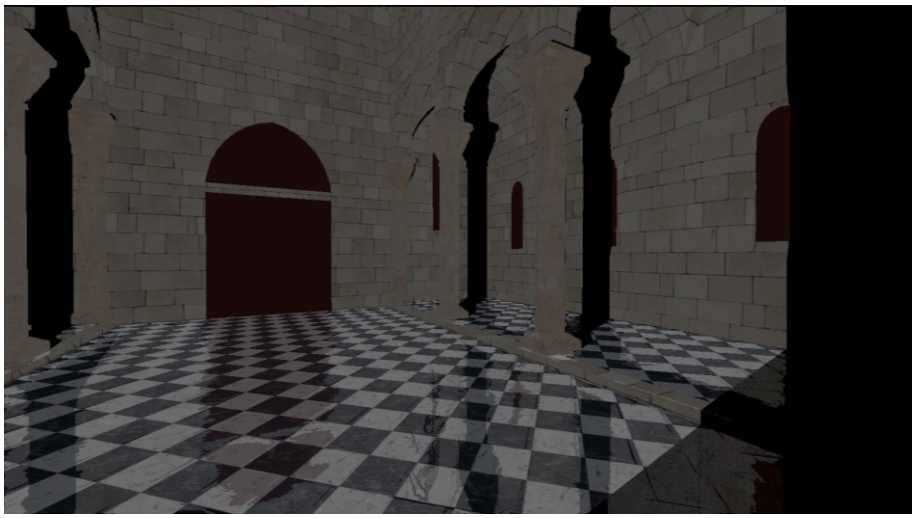


Abbildung 54: Reflektionen mit Warping und Normalmapping

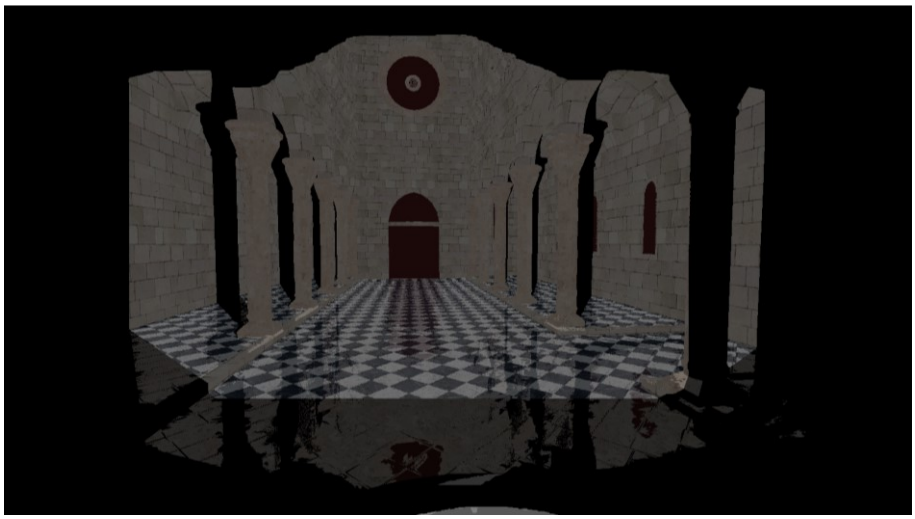


Abbildung 55: Reflektionen mit Warping und Normalmapping

5 Praxis

Um die Auswirkungen der Latenz auf die Benutzbarkeit und den subjektiven Eindruck einer Anwendung zu untersuchen, wird ein zusätzlicher Test hinzugezogen. Eine Testumgebung, in der die Latenz eine ganz entscheidende Rolle spielt ist die Oculus Rift®. Dabei handelt es sich um eine moderne VR[12]-Brille oder auch Head-Mounted-Display, die über einen Lagesensor die Kopfbewegungen des Benutzers auf die virtuelle Kamera der Anwendung übertragen kann. Zusätzliche stereoskopische Sicht soll den Immersionseffekt vergrößern.

Abbildung 56: Oculus Rift (Bildquelle: http://www.oculusvr.com/rift/)

Von Yao et al. [18] wird die Problematik der Latenz und der daraus resultierenden Simulator Krankheit (simulator sickness) näher beschrieben. Dieser Effekt entsteht durch die Diskrepanz zwischen einer visuell wahrgenommene Bewegung und der mangelnden Wahrnehmung einer tatsächlichen Bewegung. Die Symptome wirken sich in variablen Ausprägungen von Orientierungslosigkeit, Schwindelgefühl und Übelkeit aus. Die auftretenden Beschwerden führen zu einer massiven qualitativen Minderung der Anwendungen, welche die Benutzbarkeit einschränkt, weswegen die Ursachen dieses Effektes zu minimieren sind. Zum Ursprung des Effektes existieren bislang zwei Theorien. Die erste Theorie geht zurück auf die evolutionär bedingte Reaktion auf Gifte, die ebenso Wahrnehmungsstörungen hervorrufen. Der Körper versucht über Übelkeit diese Gifte wieder loszuwerden. Die zweite Theorie besagt, dass durch die widersprüchlichen In-

[12] Virtuelle Realität

formationen der Wahrnehmungsorgane die Strategien der Haltungskontrolle versagen. Die Simulator Sickness oder Motion Sickness tritt erst nach einer gewissen Zeitspanne auf.

Für die Integration der 3D-Brille wird das offizielle SDK von OculusVR genutzt. Über dieses lassen sich die Sensordaten der Brillenbewegnung direkt in Quaternionen auslesen und auf die Kamera der Anwendung übertragen. Für den stereoskopischen Effekt wird die Szene aus zwei Blickpunkten gerendert und in separate Buffer gespeichert. Jedes Bild wird unabhängig voneinander gewarpt und in der Ausgabe nebeneinander dargestellt. Zusätzlich werden die Bilder radial verzerrt.

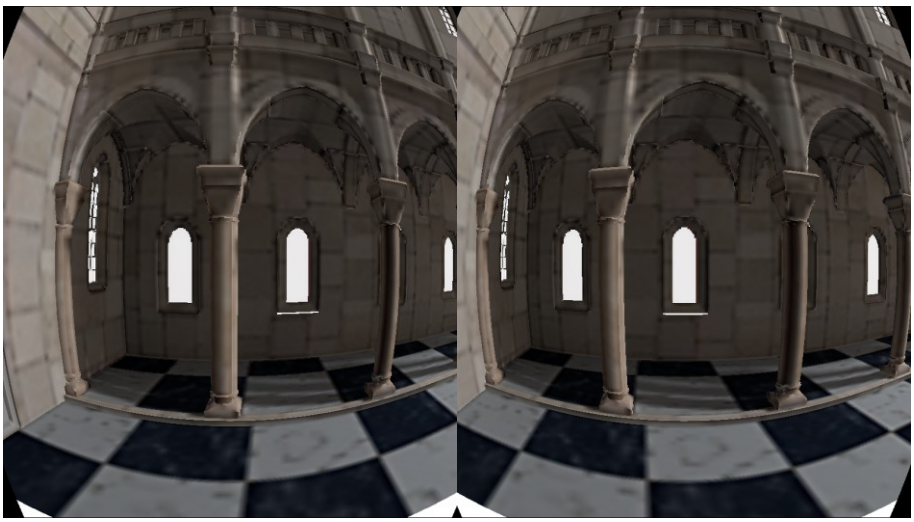


Abbildung 57: Ansicht in der Oculus Rift

5.1 **Tests**

In einem Test sollen Benutzer anhand einer subjektiven Einschätzung eine Bewertung für die Benutzerinteraktion einer Anwendung geben. Den Benutzern wird zu Beginn nach dem Aufsetzen der Brille die Anwendung ohne jegliche Verzögerung gezeigt. Das soll den Probanden zeigen, wie das HMD funktioniert und wie die Szene aussieht. Die Benutzer sollen so in den weiteren Testphasen objektiver auf die eigentliche Problemstellung des Tests eingehen und nicht die Eigenschaften der Szene oder des Gerätes bewerten. In den weiteren Stationen werden verschiedene Latenzen eingestellt. Die Benutzer bekommen wie bei einem Image Imposter RRS die komplette Latenz zu spüren. Anschließend wird das 3D-Warping aktiviert. Nach jeder Station gibt der Benutzer eine Bewertung im Bereich von 1 (sehr gut) bis 6 (ungenügend) ab. Den Benutzern wurde

kein Zeitlimit gesetzt und es war gestattet jederzeit den Text zu unterbrechen wenn diese von der Simulator Krankheit betroffen waren.

In einer ersten Testrunde sollen die Benutzer mit den unbehandelten Aufdeckungsartefakten des 3D-Warpings konfrontiert werden. Hierfür wird allein die Interpolation aktiviert. Dies soll die Möglichkeit zum Vergleich zu den Verbesserungen durch das Compositing eröffnen. Zudem stellt es eine Art „worst-case scenario“ dar. In den Tests werden die Latenzen von 200ms (Test 1,4), 400ms (Test 2,5) und 800ms (Test 3, 6) eingestellt. Tests 4-6 enthalten das aktivierte Warping.

Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**
3	4	5	3.25	4.25	-
2.25	4	5	3	2.75	3.75
3	4	5	2	2	4.25
3.5	4	5	2.25	3	4
2	5	5	3	3	3
4	4.25	5	3	3.25	3.25
3,5	4	4.25	2	2.75	3
1,5	4,5	6	2	2	2.75
5	-	-	4	-	-
3	4	4	3	3.25	3.5
2	3	4.25	2.15	3.25	3.75
2.25	5	5	1.75	2	3

Tabelle 2: Testdurchlauf 1

Die Tendenzen der ersten Testrunde zeigen, dass vor allem im Vergleich der Tests 2 und 5, also bei 400ms Latenz, ein Gewinn durch das Warping zu erwarten ist. Viele Benutzer gaben für Test 1 (200ms) an, dass die auftretende Verzögerung als wenig störend, den entstehenden Artefakten des Warpings entgegen, empfunden wurde. Ab einer Latenz von 400ms gaben alle Nutzer an die Symptome der Simulator Krankheit zu verspüren. Im Fall des ersten Testkandidaten werden diese allerdings so akut, dass der Test komplett abgebrochen werden musste. Zwei weitere Tester mussten den Test kurz unterbrechen. Im Testfall 6 wurden die Artefakte des Warpings, insbesondere bei Kamerarotationen als sehr negativ bewertet, weswegen der Unterschied zu den Bewertungen zu Test 3

geringer ausfällt. Test 3 wurde von allen Testern als kaum bis gar nicht benutzbar bewertet.

In der zweiten Testphase wird das Compositing hinzugezogen. Hierbei stiegen die Verzögerungszeiten leicht an.

Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**
2.5	4	5	2	2.5	4
3.25	6	6	4	5	4
3	5	6	4.5	6	5
3	3.5	3	1.5	2.5	3
1.5	4.5	6	4	3	3
5	-	-	3	-	-
3	4	4	4	4.5	3.25
2	3	4.25	3.25	4.25	4.75
2	5	5	4	4	3

Tabelle 3: Testdurchlauf 2

In den Ergebnissen der zweiten Testphase zeigt sich eine ähnliche Verteilung der Bewertungen. Oftmals fielen die Bewertungen in diesen Fällen schlechter aus gegenüber dem Warping ohne Compositing. Die Tester geben an, dass diese durch ein verstärktes Flimmern gestört wurden.

5.2 **Bewertung**

Die durchschnittlichen Bewertungen innerhalb des ersten Testdurchlaufens zeigen ein besseres Abschneiden des Warpings. Bei einer Verzögerungszeit von 200ms wird das Warpingverfahren um 0.4 Punkte, bei 400ms um 1,35 und bei 600ms um 1,4 Punkte besser bewertet. Der Unterschied steigt mit Zunahme der Latenz. Daraus lässt sich schließen, dass sich ein Gewinn aus dem Verfahren ziehen lässt. Für den zweiten Testdurchlauf fallen die Ergebnisse mit einem geringeren Unterschied aus. Bei einer Verzögerungszeit von 200ms liegt das Warpingverfahren um -0.5 Punkte, bei 400ms um 0,475 und bei 600ms um 1,15 Punkte besser. Als Fazit lässt sich daraus schließen, dass das Verfahren für eine Verwendung noch weiter verbessert werden muss.

Einige Probanden geben an, dass das Verfahren helfe die Simulator Krankheit zu reduzieren. Zugleich kritisierten sie die Artefakte und gaben dafür eine schlechtere Bewer-

tung ab. Während der Tests hat sich gezeigt, dass die Ergebnisse für Tests mit längeren Latenzen besser ausfielen, wenn die Probanden das Gerät vorher abgenommen haben. Dies lässt darauf schließen, dass die Tester teilweise verzögerte Symptome der Simulator Krankheit mit in folgende Tests übertrugen, auch wenn diese angaben keine Symptome zu verspüren.

Grundlegend fielen die Bewertungen generell sehr schlecht aus. Viele Tester beschwerten sich über die niedrige Auflösung und erwähnten ein Rauschen oder Flimmern im Bild. Damit deuteten die Benutzer vermutlich auf die starken Aliasing Effekte der Objektkanten im Hintergrund der Szene. Insbesondere beim Warping wird dieses Problem durch die verdoppelten Objektkanten zwischen leeren Bereichen und den tatsächlichen Objekten verstärkt.

Ein weiteres Problem der Hardware ist die interne Latenz. Jeder Bildschirm benötigt eine gewisse Zeit, bis dieser das gesendete Bild darstellt. Im Falle der Oculus Rift liegt diese bei ungefähr 50 Millisekunden[13]. Für eine genaue Bestimmung der Latenz ist ein spezielles Messgerät notwendig, das für diesen Test nicht vorlag. Die Latenz kann durch das Warping nicht herausgerechnet werden. In einigen Fällen reicht diese Latenz aus, um bei Benutzern die Simulator Krankheit auszulösen. Die anfängliche Gewöhnungsphase sollte Aufschluss darüber geben. Allerdings lässt sich nicht voraussagen, ab wann genau der Effekt der Simulator Krankheit eintritt. Da diese Latenz in allen Test vorlag, kann diese einen negativen Einfluss auf die späteren Testergebnisse genommen haben.

[13] http://oculusrift-blog.com/john-carmacks-message-of-latency/682/

6 Zusammenfassung

Remote Rendering Systeme finden in der heutigen Zeit erste reale Anwendungen. Diese basieren jedoch ausschließlich auf der Übertragung des finalen Bildes und der Benutzereingaben. Die Grundlagen des Renderns und eines solches Systems wurden in Kapitel 2 erläutert. Auch weitere Grundlagen zu der auftretenden Verzögerung und dem 3D-Wapring, eine Technik zur Blickpunkt Interpolation wurden vorgestellt. Neben der direkten Übertragung aus Ausgabe des gerenderten Bildes gibt es weitere Ansätze, die nur die Geometrie oder ausschließlich die Beleuchtungsinformation übertragen.

Die Umsetzung des pixelbasierten 3D-Warpings hat gezeigt, dass es ein effizientes Verfahren ist, welches in echtzeitfähigen Programmen genutzt werden kann. Das Verfahren ist besonders für Remote Rendering Systeme geeignet, in der eine direkte Benutzerinteraktion essentiell ist. Insbesondere in einem hybriden System lässt sich das Verfahren soweit integrieren, dass einzig die Aufdeckungsproblematik in weiteren Schritten bearbeitet werden muss.

Alle Umsetzungen können die durch das Warping auftretenden Artefakte beheben oder weitestgehend reduzieren. Jedoch gilt für all diese Verfahren, dass deren Zuverlässigkeit mit Zunahme der Latenz sinkt. Da es sich bei den getesteten Latenzen um große bis sehr große Werte handelt, können diese Verfahren einen Nutzen für bisherige Remote Rendering Systeme darstellen. Einen genaueren Aufschluss über die Anwendbarkeit sollte die Nutzung der Techniken in einem Setup mit einem Head-Mounted Display und dem damit verbundenen Phänomen der Simulator Krankheit geben. Auf Grund der unzureichenden Eigenschaften der Hardware lässt sich bisher keine zuverlässige Aussage treffen.

7 Ausblick

Die Beispiele der verfügbaren Remote Rendering Systeme haben gezeigt, dass Image Imposter bereits Verwendung in echtzeitfähigen Anwendungen finden. Mit den erreichten Verzögerungen von ca. 100 Millisekunden oder weniger ist eine Interaktivität gegeben. Hingegen hat sich auch gezeigt, dass selbst lokale Netzwerke mit ausreichend Bandbreite Schwankungen nicht ausschließen können. Die Implementierung eines 3D-Warping Verfahrens in ein RRS kann nicht nur dazu dienen, die verbleibende Latenz zu reduzieren, das insbesondere für die Verwendung bei Head-Mounted-Displays interessant ist. Auch zum Ausgleich der Schwankungen des Netzwerkes und der damit einhergehenden Minderung der Interaktivität kann ein solches Verfahren eine Lösung darstellen. Die in Teil 4 dieser Arbeit vorgestellten Ansätze wurden zumeist mit der doppelten oder dreifachen Latenz untersucht. Somit stellen diese Tests Szenarien von sehr großen Verzögerungsschwankungen dar. Denn für die Verwendung des 3D Warpings gilt, dass die auftretenden Artefakte mit der Abnahme der Latenz reduziert werden.

Das Compositing Verfahren weist das Problem auf, dass die gesuchten Beleuchtungsinformationen nicht vollständig von den Texturwerten getrennt werden konnte. Für dieses Fall bleibt zu untersuchen, welchen Gewinn das Verfahren daraus ziehen kann, wenn das Referenzbild nur die Beleuchtungsinformationen beinhaltet. Für die Ausgabe ließen sich die Beleuchtungsinformationen auf das unbeleuchtete Rendering des Clients legen.

Das Verfahren von 4.7 hat gezeigt, dass sich auch Blickpunkt abhängige Effekte, wie Reflektionen, mit dem 3D-Warping Verfahren abbilden lassen, sofern für alle Punkte die Tiefen bekannt sind. Dies gilt allerdings nur für planare Spiegelungen, welcher durch Post-Processing Effekte, beispielsweise Normal Mapping erweitert werden können, wie gezeigt wurde. Für konkave oder konvexe Oberflächen kann ein SSR Verfahren zumindest eingeschränkt eine Lösung darstellen, da dies auf dem Bildbereich des Referenzbildes angewendet werden kann.

Der größte Kritikpunkt des direkten 3D-Warpings lag in der schlechteren Bildqualität, insbesondere in den lückenbehafteten Oberflächen des abgeleiteten Bildes. Die Ergebnisse der Implementationen zeigen, dass bis zu einer bestimmten Verzögerungszeit eine einfache Filterung ausreicht, um diese Lücken zu schließen. Diese Lösung ist abhängig von den Vorgaben der Anwendung bezüglich der Kamerasteuerung. Ein direkter Vergleich zu dem meshbasierten Warpingverfahren nach [9] ist interessant, das Interpolationsfehler durch die Rasterung eines Fragmentprogramms unterdrückt. Das kann im Vergleich zu den Limitierungen der Kamerasteuerung und der resultierenden Bildquali-

tät des pixelbasierten Verfahrens gesetzt werden. Wie bereits erwähnt müssen durch die „Rubber Sheets“ überdeckten, leeren Bereiche im meshbasierten Verfahren für ein späteres Compositing extrahiert werden. Ein möglicher Ansatz liegt darin, den Vertices vor dem Warping eindeutige IDs zuzuweisen. Nach der Rasterung sind alle Bildpunkte mit veränderten IDs jene, die vom Grafikprozessor interpoliert wurden. Besitzt ein Pixel viele Nachbarn mit interpolierten Werten, ist ein „Rubber Sheet“ gefunden.

Die Verwendung von Compute Shader bietet eine einfache und performante Lösung um ein 3D-Warping auf einem Bild durchzuführen. In Bezug auf die Problematik des Tiefentests und die damit einhergehende doppelte Berechnung der neuen Koordinaten erscheint die Verwendung der Shader als unpraktikabel. In dem hybriden Aufbau des RSS werden die Tiefenwerte auf dem Client berechnet. Somit kann der Tiefentest direkt im ersten Shaderpass durchgeführt werden, welches in Bezug auf die Tiefe eine optimale Qualität bietet. Bei dem Warping werden Bildkoordinaten nicht auf exakte Pixelkoordinaten abgebildet. Die Benutzung von Splats sollte alle Bildpunkte, die von dem abgebildeten Pixel berührt werden, mit dem prozentualen Anteil ihres Farbwertes ergänzen, wodurch die Präzision des Warpings verbessert würde. Es gilt hierfür eine adäquate Lösung zu finden.

Erste Präsentationen und Pressemitteilungen verraten eine Weiterentwicklung und Verbesserungen der Oculus Rift VR-Brille. So wird nun auch die Körperhaltung mit in die Kamerasteuerung integriert, um den Immersion Effekt weiter zu verstärken. Die interne verzögerte Bildausgabe ist nach Hersteller Angaben minimiert. Auch die massive Problematik der niedrigen Auflösung wird behandelt. In einer neu vorgestellten Version wird die Anzahl der Bildpunkte mehr als verdoppelt. Eine Wiederholung des Testes aus der Arbeit wäre sehr interessant, da die Beurteilungen der Benutzer weniger stark durch die Schwächen des Gerätes und die damit geschmälerte Qualität weniger beeinflusst würden.

In dem konkreten Testszenario handelt es sich ausschließlich um eine statische Szene in welcher keinerlei Interaktion mit Gegenständen gegeben ist. Die angesprochene Erweiterung des Testes in eine dynamische Szene würde weitere Forschungsfragen aufwerfen, da in dieser Arbeit keine animierten Objekte untersucht werden konnten. Hierbei sind zwei verschiedene Arten von dynamischen Objekten zu untersuchen. Festkörper (rigit bodies) werden über ihre Position, Rotation und Skalierung animiert. Es muss eine Lösung analysiert werden, diese Objekte für ihre individuellen Animationen zu warpen und auftretende Artefakte zu beheben. Hierfür gilt zu bestimmen welche vom Server gesendeten Pixel dem bewegtem Objekt zugehörig sind. Dies könnte über ein separates

Rendering auf dem Client, welchem die Geometrie ebenso vorliegt zu realisieren sein. Anschließend können die Bildpunkte analog zu dem vorgestellten Verfahren mit der Modelmatrix des Objektes gewarpt werden. Für Weichkörper gilt dies hingegen nicht, da sich innerhalb dieser weitere Bewegungen abspielen. Da sich jedes primitive des Objektes individuell bewegen kann, müssen diese Transformationen auf dem Client berechnet werden. Ein Compositing könnte die Beleuchtung des Objektes erneuern.

Zu einer weiteren Verbesserung der Testergebnisse, die zu einer präziseren Aussage zum Einsatz des Warpingverfahren führen kann, ist die Übertragung in eine zielgerichtete Anwendung. Hierbei könnte es sich beispielsweise um eine komplexere, virtuelle Welt oder ein Spiel handeln. Der Benutzer könnte die Erfüllung der darin gestellten Aufgabe in direktem Zusammenhang mit der, durch die Latenzen oder Artefakte des Warpings, verstärkten Erschwerung setzen und bewerten. Die Bewertungen können so objektiver ausfallen und wären weniger von der Beschreibung des Testleiters abhängig. In einem vollständigen RRS lässt sich zusätzlich die Netzwerkauslastung unter Hinzuziehung von mehreren Bildern für das Warping von Reflektionen untersuchen. Das ist insofern interessant, da das Bild mit den Reflektionsinformationen nicht der Größe des Ausgangbildes, sondern nur der Bereich, in dem die Reflektionen auftreten, entsprechen muss. Des Weiteren muss für jede Rekursionsstufe der Reflektionen ein neues Bild an den Client gesendet werden. In einem RSS kann das komplette Verfahren weitere Clientsysteme mit unterschiedlichen Leistungsstufen untersucht werden.

8 Abbildungsverzeichnis

9 Literaturverzeichnis

[1] AFP, „stern.de,“ 14 08 2013. [Online]. Available: http://www.stern.de/news2/aktuell/weltweit-mehr-smartphones-als-normale-handys-verkauft-2050974.html.

[2] DPA, „faz.de/wirtschaft,“ 04 04 2013. [Online]. Available: http://www.faz.net/aktuell/wirtschaft/computer-bald-mehr-tablets-als-pcs-und-laptops-12137270.html.

[3] C. Crassin, D. Luebke, M. Mara, M. McGuire, B. Oster, P. Shirley, P.-P. Sloan und C. Wyman, „CloudLight: A system for amortizing indirect lighting in real-time rendering,“ NVIDIA, 2013.

[4] S. Shi, „Low Latency Remote Rendering System for Interactive Mobile Graphics,“ Urbana, Illinois, 2012.

[5] P. Eisert und P. Fechteler, „Remote rendering of computer games,“ Fraunhofer Institute for Telecommunications, Berlin, 2007.

[6] NVIDIA Corporation, „video capture, endocing and streaming in multi-gpu system,“ NVIDIA Corporation, 2010.

[7] H. Zheng, E. K. Lua, M. Plas und T. G. Griffin, „Internet Routing Policies and Round-Trip-Times,“ University of Cambridge Computer Laboratory.

[8] L. McMillian und G. Bishop, „Head-tracked stereographic display using image warping,“ Sitterson Hall, Chapel Hill, NC 27599, 1995.

[9] W. R. Mark, „Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping,“ Chapel Hill, NC 27599-3175, 1999.

[10] G. Schmidt, „Rendering view dependent reflections using the graphics card,“ Koblenz, 2013.

[11] S. N. Sinha , J. Kopf, M. Goesele, D. Scharstein und R. Szeliski, „Image-Based Rendering for Scenes with Reflections“.

[12] J. Leech, P. Womack und P. Karlton, OpenGL® Graphics with the X Window System®, Silicon Graphics, Inc., 2005.

[13] M. Levoy, „Polygon-Assisted JPEG and MPEG Compression of Synthetic Images," 1995.

[14] G. Drettakis und F. Duguet, „Flexible Point-Based Rendering on Mobile Devices," 2004.

[15] W. R. Mark, L. McMillan und G. Bishop, „Post-Rendering 3D Warping," 1997.

[16] W. Yoo, S. Shi, W. J. Jeon, K. Nahrstedt und R. H. Campbell, „real-time parallel remote rendering for mobile devices using graphics processing unit," Urbana, IL, 61801, USA, 2010.

[17] E. Walia und V. Verma, „A Computationally Efficient Framework for 3D Warping Technique," International Journal of Computer Graphics Vol. 3, No. 1, New Delhi, 2012.

[18] R. Yao, T. Heath, A. Davies, T. Forsyth, N. Mitchell und P. Hoberman, Oculus VR Best Practices Guide, Oculus VR, Inc., 2014.

[19] G. B. Lochmann und H.-C. Wollert, Autoren, *Echtzeit Rendering.* [Performance]. Universität Koblenz-Landau, WS 13/14.

[20] S. Shi, K. Nahrstedt und R. Campbell, „A Real-Time Remote Rendering System for Interactive Mobile Graphics," ACM Trans. Multimedia Comput. Commun, 2012.