UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Optimierung des Reflective-Warping Verfahrens

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Moritz Löhne

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Gerrit Lochmann Deutsches Institut für Normung, Abteilung Moderne Technologien

Koblenz, im Mai 2015

Erklärung

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

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

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

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 Bachelorarbeit
Moritz Löhne
(Mat. Nr. 211100467)

Thema: Optimierung des Reflective-Warping-Verfahrens

Die Erzeugung neuer Kamerapositionen in vorhandenen Tiefenbildern ist ein Verfahren, das unterschiedliche Verwendungszwecke erfüllt: Etwa bei der Erzeugung von Zwischenbildern, zur Latenzreduktion in einem Remote Rendering Setup oder zur Stereo-Bildsynthese. Das so genannte Warping liefert jedoch nur bei diffusen Oberflächen valide Ergebnisse. Sobald optische Effekte, wie Transparenz oder Spiegelung, auftauchen, ist die Zerlegung des Bildes in Ebenen nach dem Grad der Lichtindirektion erforderlich. Im Fall der ersten Lichtindirektion wird das Warping durch ein Optimierungsverfahren gelöst, das bei hohen Bildfrequenzen zu Artefakten führt.

In dieser Arbeit soll ein Raytracer bzw. Pathtracer entwickelt werden, der diese Aufteilung beim Rendervorgang vornimmt und den Input für das Layered Warping liefert. Zusätzlich sollen Optimierungen vorgenommen werden, die insbesondere für die erste Indirektion gleichmäßigere Ergebnisse niedriger Frequenzen liefern und das Warping-Verfahren unterstützt. Beim Output sollen die hohen Frequenzen durch einfaches Normalmapping nachträglich rekonstruiert werden.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche über Ray- /bzw. Pathtracing
2. Recherche über das reflektive Warping-Verfahren
3. Implementierung eines prototypischen Ray- / Pathtracers
4. Implementierung des reflective warping Algorithmus
5. Optimierung des Verfahrens durch Glättung
6. Evaluation hinsichtlich der visuellen Qualität und der Auswirkungen auf das Optimierungsproblem
7. Dokumentation der Ergebnisse

Koblenz, 21.10.2014

– Moritz Löhne –

– Prof. Dr. Stefan Müller –

Inhaltsverzeichnis

Abbildungsverzeichnis

Zusammenfassung

Thematik dieser Arbeit ist das dreidimensionale Image-Warping für diffuse und reflektierende Oberflächen. Das Warpingverfahren für den reflektierenden Fall gibt es erst seit 2014. Bei diesem neuen Algorithmus treten Artefakte auf, sobald ein Bild für einen alternativen Blickwinkel auf eine sehr unebene Fläche berechnet werden soll. In dieser Arbeit wird der Weg von einem Raytracer, der die Eingabetexturen erzeugt, über das Warpingverfahren für beide Arten der Oberflächen, bis zur Optimierung des Reflective-Warping Verfahrens erarbeitet. Schließlich werden die Ergebnisse der Optimierung bewertet und in den aktuellen, sowie zukünftigen Stand der Technik eingeordnet.

Abstract

This thesis broaches the issue of threedimensional Image-Warping for diffuse and reflective surfaces. The procedure of the reflective case exists since 2014. Within this new algorithm artefacts appear, once a picture with a user-given perspective for a rough surface is computed. This thesis comprises a raytracer which produces textures for the warping-algorithm. Further the warping for both cases of surfaces and its optimization are covered. Last, the results of the optimization are evalued and its possible view in future technique is given.

1 Einleitung

Eine dreidimensionale Sicht, Head-Mounted Displays oder Remote-Rendering Systeme werden immer häufiger für die Präsentation computergenerierter Bilder benutzt. Zur Stabilisation der Bildfrequenz oder zum Stereo-Rendering ist das Reflective-Warping Verfahren von Vorteil.
Mit der Eingabe von einzelnen Tiefenbildern ist es möglich, Bilder für einen beliebigen Blickwinkel auf die Szene neu zu berechnen. Dabei ist es nicht notwendig, die Bilder aufwändig neu zu rendern; Berechnungen ermitteln Position und Farbe für das neue Bild. High End Hardware ist dafür nicht gefordert, wodurch das Verfahren gerade für Remote-Rendering Systeme von Bedeutung ist.
Das Verfahren des Image Warpings besteht bereits seit über 15 Jahren. Allerdings war es bisher nicht möglich Reflexionen und Lichtbrechungen aus einem frei gewählten Blickwinkel betrachten zu können, ohne die Szene komplett neu rendern zu müssen. Der Algorithmus, der dies nun ermöglicht, wird in dieser Arbeit vorgestellt. Außerdem wird das Reflective-Warping optimiert und die daraus folgenden Ergebnisse werden hinterfragt und bewertet.
Die Thematik dieser Arbeit reicht von der Synthese der Eingabebilder bis zur Optimierung des Verfahrens. Für unebene Oberflächen mit deutlich verschiedenen Ausrichtungen der Normalen entstehen Artefakt im Ausgabebild. Das Ziel der Arbeit ist, der Weg zu diesem Problem und die Bewertung einer Lösung. Bei erfolgreicher Problemlösung wird ein stabilerer Algorithmus erreicht, der zukünftig in vielen Anwendungsgebieten zum Einsatz kommen kann.
Für die Generierung der Eingabebilder wird ein Raytracealgorithmus benutzt, der auf das Notwendigste beschränkt wird. Anschließend erfolgt das Warping für den diffusen und den reflektierenden Fall und die Zusammenfügung dieser. Zum Schluss wird eine Lösung für die oben genannte Problematik erarbeitet und bewertet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für das Reflektive-Warping-Verfahren erläutert. Es wird auf den Aufbau und die Funktionsweise eines Ray- bzw. Pathtracers eingegangen. Dieser Algorithmus erzeugt die Eingabebilder für das im Mittelpunkt stehende Reflective-Warping. Von dem Raytracealgorithmus wird nur das Wesentliche erläutert und nur einige Erweiterungen werden genannt.
Anschließend wird das Verfahren des Warpings, sowohl für den diffusen, als auch für den reflektierenden Fall, ausgeführt. Am Schluss des Kapitels wird erklärt, wie die vorher getrennten Fälle wieder zusammengeführt werden.
Für diese Arbeit werden Grundkenntnisse der Grafikprogrammierung und technische Begriffe, die damit zusammenhängen vorausgesetzt. Ebenso werden manche englischen Begriffe nicht übersetzt, da sie bezeichnender wirken als ihre Übersetzung.

2.1 Ray- und Path Tracing

Der Ray- bzw. Path Tracing-Algorithmus (*dt.: Strahl- / Pfadverfolgung*) ermöglicht das Simulieren einer virtuellen Szene mit physikalisch korrektem Verhalten von Licht [Hen11, S. 9]. Beispielsweise können weiche Schatten und eine globale Beleuchtung erreicht werden, wodurch das computergenerierte Bild überzeugender wird.

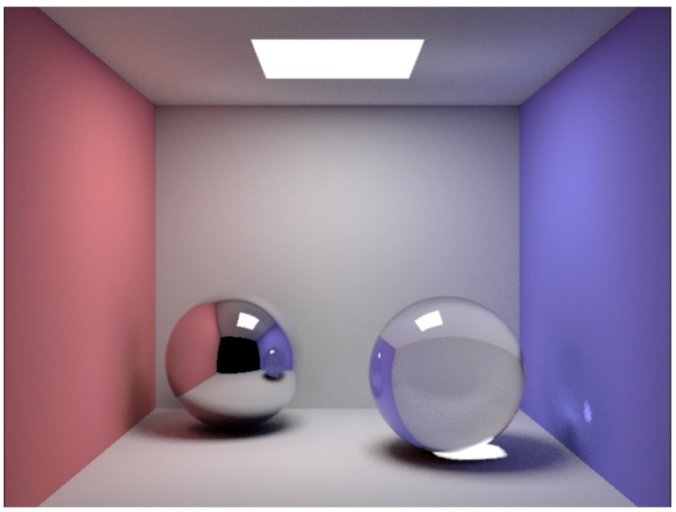


Abbildung 1: Ein mit Path Tracing erstelltes Bild

http://www.ccs.neu.edu/home/fell/images/CSG140/HW/presentationsSP2009/Vyas-Global-Illumination.jpg

Eine globale Beleuchtung wird anhand Abbildung 1 deutlich. Die Farbgebung der Wände wird von der Farbe angrenzender Objekte beeinflusst, was als *Color Bleeding* bezeichnet wird und an den oberen hinteren Ecken

beobachtet werden kann. Weiche Schatten sind in diesem Bild außerdem zu erkennen. Der Übergang von schwarzem Schatten zur Farbe auf dem Boden ist unter der rechten Kugel deutlich erkennbar.
1968 stellte Arthur Appel bereits einen rudimentären Raytracer vor [App68], während James Kajiya den Path Tracing Algorithmus 1986 veröffentlichte [Kaj86]. Die beiden Begriffe, Raytracing und Path Tracing, werden im Folgenden vorerst als Synonym behandelt und später unterschieden, da beide Verfahren ähnlich aufgebaut sind, aber verschiedene Vor- und Nachteile haben. Aus diesen unterschiedlichen Eigenschaften ergeben sich verschiedene Einsatzmöglichkeiten.
Anhand Abbildung 2 wird der Aufbau dargestellt [Abe09, S. 20]:

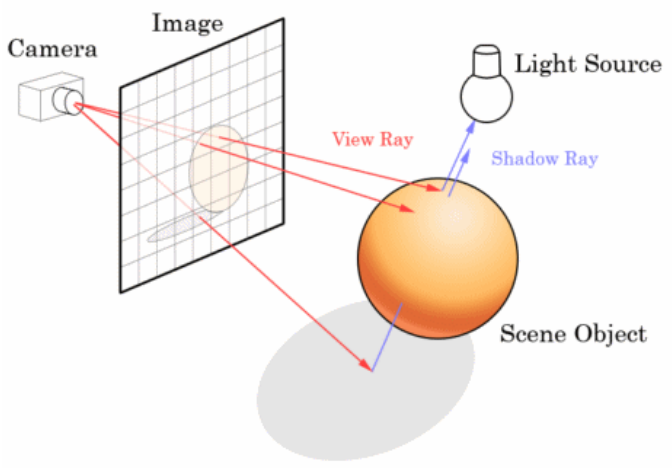


Abbildung 2: Aufbau eines Raytracers
http://blog.codinghorror.com/real-time-raytracing/

Von einem definierten Punkt in der Szene, dem Augpunkt, der vergleichbar mit einer Kamera ist, werden Strahlen durch eine Bildebene in die Szene geschickt. Diese sogenannten *Primärstrahlen* ermitteln Schnittpunkte mit Geometrie der Szene und berechnen einen Farbwert für die getroffene Oberfläche. Die Punkte der Bildebene, durch die die Strahlen verlaufen, entsprechen den Pixeln des Ausgabebildes.
Der Strahl ist ein Vektor und berechnet sich aus dem Augpunkt und der aktuellen Position auf der Bildebene. Dieser Vektor wird mit einem Skalar multipliziert, der die Länge des Strahls bestimmt. Die Position der Geometrie wird dadurch als Vektor dargestellt. Die Gleichung dafür sieht wie folgt aus:

$$\vec{p} = c + t\boldsymbol{P}x \tag{1}$$

Hierbei stellt der Vektor $\vec{p}$ den getroffenen Punkt in der Szene dar. c steht für den Augpunkt, aus dem der Strahl ausgeht, $\boldsymbol{P}x$ berechnet die Richtung des Vektors und t skaliert ihn. x besteht in dieser Gleichung aus den Ko-

ordinaten der Bildebene. Detaillierter wird die Gleichung im Abschnitt des Warpings beschrieben.
Jeder in die Szene geschickter Strahl muss mit jedem Objekt auf einen Schnittpunkt überprüft werden. Dabei werden Strahl und Objekt in einer Methode parametrisiert und die kürzeste Entfernung zum Augpunkt als Ergebnis zurückgeliefert, wodurch dem Betrachter immer das dichteste Objekt erscheint. Die Verdeckung ist berechnet. Dadurch liegen nicht nur dreidimensionale Koordinaten, sondern auch die Tiefe für die Geometrie in der Szene vor, was für das Warping von entscheidender Bedeutung ist. Effizient berechenbare Objekte sind bspw. Primitive wie das Dreieck, die Kugel oder der Zylinder, da diese mit wenigen Parametern reproduzierbar sind. Sobald ein Objekt getroffen wird, wird die Farbe an der entsprechenden Position ermittelt und *Sekundärstrahlen* entstehen, die wiederum in die Szene geschickt werden. Das Entstehen von Sekundärstrahlung gilt bereits als Erweiterung des klassischen Raytracers. Dieser bricht bei erstem Schnittpunkt und Berechnung der Farbe ab [App68].
Die Brechung, sowie die Reflexion des Strahls, werden als Sekundärstrahlen zusammengefasst. Die Berechnung des Reflexionsstrahls sieht folgendermaßen aus und wird in Abbildung 3 dargestellt:

$$\vec{r} = 2(\vec{n} \cdot \vec{\Psi})\vec{n} - \vec{\Psi} \tag{2}$$

Um den reflektierten Strahlenvektor $\vec{r}$ zu ermitteln, wird die Normale $\vec{n}$ an der geschnittenen Position der Oberfläche benötigt. Diese beschreibt die Ausrichtung der Fläche, wodurch der Austrittsvektor berechnet wird [DBB06, S. 36f]. Es gilt: Eintritts- gleich Austrittswinkel. $\vec{\Psi}$ ist der einfallende Primärstrahl.

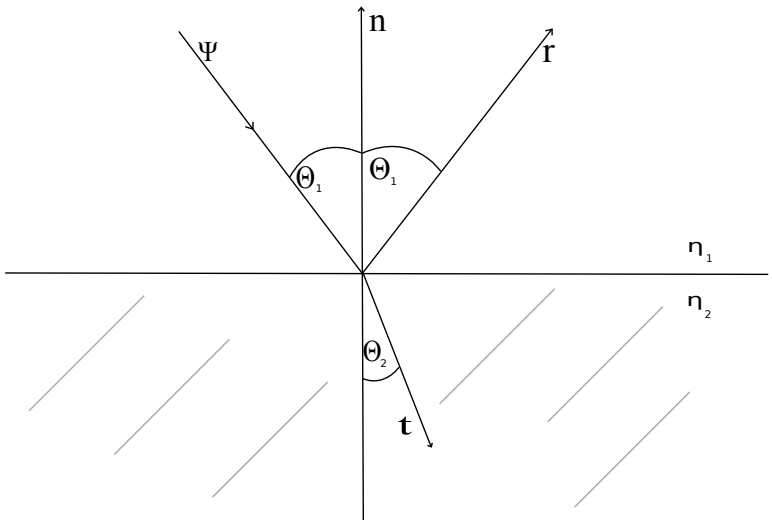


Abbildung 3: Reflexion und Brechung an einer spiegelnden Oberfläche

Erst wenn ein Abbruchkriterium erfüllt ist, wird der Strahl nicht weiter reflektiert, die finale Farbe steht fest. Wie die Farbe einer Oberfläche erscheint, bestimmt das gewählte Beleuchtungsmodell und das Shading. Der Blickwinkel der Kamera, die Normale der Oberfläche und der Winkel zu der Lichtquelle werden dazu benötigt. Es wird berechnet, wie viel Licht auf die Oberfläche fällt und wie gesättigt die Farbe dem Betrachter angezeigt wird.

Auch die Brechung des Lichts ist korrekt simulierbar. Das Verhalten vom gebrochenen Anteil des Strahls wird durch das *Snelliussche Brechungsgesetz* beschrieben [DBB06, S. 36]:

$$\eta_1 \sin(\delta_1) = \eta_2 \sin(\delta_2) \tag{3}$$

η_1 und η_2 stehen für die Brechungsindizes der durchlaufenen Materie. Sie geben an, wie stark der Strahl gebrochen wird. δ_1 ist der Eintrittswinkel, δ_2 ist der gesuchte Austrittswinkel.
Mit dem Snelliuschen Brechungsgesetz und den Parametern des einfallenden Strahls, kann der gebrochene Strahl komplett berechnet werden [DBB06, S. 36f]:

$$\begin{aligned} \vec{t} &= -\frac{\eta_1}{\eta_2}\vec{\Psi} + \vec{n}\left(\frac{\eta_1}{\eta_2}\cos\theta_1 - \sqrt{1-(\frac{\eta_1}{\eta_2})^2(1-\cos\theta_1^2)}\right) \\ &= -\frac{\eta_1}{\eta_2}\vec{\Psi} + \vec{n}\left(\frac{\eta_1}{\eta_2}(\vec{n}\cdot\vec{\Psi}) - \sqrt{1-(\frac{\eta_1}{\eta_2})^2(1-(\vec{n}\cdot\vec{\Psi})^2)}\right) \end{aligned} \tag{4}$$

$\vec{t}$ ist der resultierende gebrochene Sekundärstrahl, ebenfalls visualisiert in Abbildung 3. Durch die Definition des Skalarprodukts ist die Umformung $\cos\theta_1 = \vec{n}\cdot\vec{\Psi}$ möglich. $\vec{\Psi}$ ist der Vektor des Primästrahls und $\vec{n}$ die Normale der Oberfläche. Da nun der gebrochene und der reflektierte Strahl vorliegen, muss entschieden werden, wie groß jeweils der Anteil des Sekundärstrahls ist.
Mit den *Fresnelschen Formeln* kann der Reflexionsfaktor berechnet werden, wodurch die Gewichtung der entstehenden Sekundärstrahlen ermittelt wird [DBB06, S. 37]:

$$\begin{aligned} \rho &= \frac{1}{2}\cdot(\rho_\parallel^2 + \rho_\perp^2) \\ \rho_\parallel &= \frac{\eta_2\cos(\theta_1) - \eta_1\cos(\theta_2)}{\eta_2\cos(\theta_1) + n_1\cos(\theta_2)} \\ \rho_\perp &= \frac{\eta_1\cos(\theta_1) - \eta_2\cos(\theta_2)}{\eta_2\cos(\theta_1) + \eta_1\cos(\theta_2)} \end{aligned} \tag{5}$$

Die beiden Summanden der Gleichung 5 stehen für die Polarisation des Lichtstrahls, die Richtung der Schwingung. $\rho_\parallel$ drückt den Anteil der Richtung der Schwingung aus, der parallel zur Einfallsebene verläuft. $\rho_\perp$ den Anteil, der senkrecht zur Einfallsebene steht. Visualisiert wird die Aufteilung

in Abbildung 4. η_1 und η_2 stehen für die Brechungsindizes der Materialien. θ_1 für den Eintritts- und θ_2 für den Austrittswinkel. ρ gibt den reflektierten Anteil des Strahls wider.

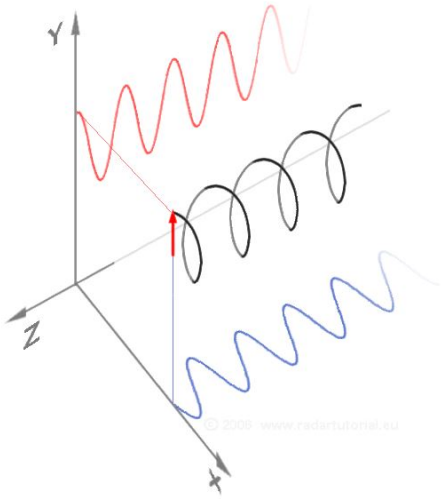


Abbildung 4: Parallele und senkrechte Polarisation

http://www.3dstuff.org/images/textseite_foto_zirkular.gif

Auch Schatten sind beim Raytracing möglich [App68], gelten aber als Erweiterung des klassischen Algorithmus. Dafür wird von einem beliebigen Punkt ein Strahl in Richtung der Lichtquelle geschickt und auf einen Schnittpunkt mit Geometrie getestet. An der Ausgangsposition kann Schatten gesetzt werden, sobald ein Objekt getroffen ist. Dieser Strahl wird *Schattenfühler* genannt. Solange mit nur einem Schattenfühler gearbeitet wird, sind nur harte Schatten möglich bzw. entsteht Bildrauschen. Werden mehrere Strahlen von einem Punkt in zufällige, verschiedene Richtungen ausgesandt und entsprechend gemittelt, ist es möglich, weiche Schatten darzustellen, die der Realität näher sind [DBB06, S. 118ff]. Abbildung 5 verdeutlicht dies.

Obwohl mit dem Raytracealgorithmus optisch hochwertige Bilder erzeugt werden können, ist das Verhalten des Lichtes nicht physikalisch korrekt. Bei dem Path Tracing Algorithmus wird hingegen die Rendergleichung [Kaj86] gelöst. Dieses Verfahren ist physikalisch fundierter und soll nun in groben Zügen erläutert werden.
Die bisher beschriebenen Verfahren liefern keine globale Beleuchtung, da Materialien nur direkt beleuchtet werden und keine Strahlen von diffusen Oberflächen reflektiert werden. Bei dem Path Tracing werden diese Strahlen geschickt, wodurch Effekte wie Color bleeding und Kaustiken ermöglicht werden [Abe09, S. 208].
Grundbaustein der globalen Beleuchtung ist das Lösen der Rendergleichung. Bei der Gleichung wird von globalen Strahlenbündeln ausgegangen, die von

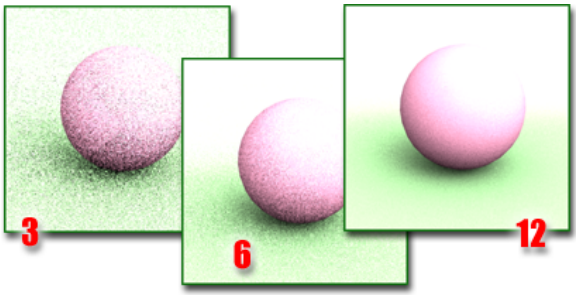


Abbildung 5: Bildrauschen bei zu wenig Schattenfühlern

http://help.autodesk.com/cloudhelp/2015/ENU/3DSMax/images/

GUID-8E25E16F-15BA-4686-8E4F-674BC0FE9B82.png

der Lichtquelle durch die Szene geschickt werden. Jede Oberfläche hat somit das Potenzial beliebig viele Strahlen zu empfangen. Mit der Rendergleichung wird die Leuchtdichte auf der Fläche x mit Reflexionsrichtung $\vec{\omega}_o$ berechnet [PH10]:

$$L(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \int_{\Omega} L_i(x, \vec{\omega}_i) f(x, \vec{\omega}_i, \vec{\omega}_o) \cos\theta d\vec{\omega}_i \tag{6}$$

Das Integral umfasst die Richtungen, aus denen Licht auf die Fläche x fallen kann. Wie in Abb. 6 dargestellt, erfüllt die Halbkugel über der Fläche x diese Forderung. Die Lösung des Integrals kann mit Monte-Carlo-Methoden vereinfacht werden [Hon13]. $\vec{\omega}_i$ stellt die Richtungen dar, aus der das Lichtbündel einfällt. $\vec{\omega}_o$ die Richtung der Reflexion. Falls es sich bei der zu berechnenden Fläche um eine Lichtquelle handelt, berechnet $L_e(x, \vec{\omega}_o)$ die emittierte Strahlung. $f(x, \vec{\omega}_i, \vec{\omega}_o)$ beschreibt die Bidirektionale Reflexionsverteilung, die *BRDF*. Sie berechnet das Reflexionsverhalten von Oberflächen und ist damit ausschlaggebend für die optische Erscheinung von Objekten [NRH$^+$77]. Die eingehende Leuchtdichte wird mit $L_i(x, \vec{\omega}_i)$ ausgedrückt und θ gibt den Winkel zwischen Normale und der Einfallsrichtung $\vec{\omega}_i$ an.

Die Rendergleichung soll nicht weiter vertieft werden, da Path Tracing und seine Möglichkeiten nicht im Mittelpunkt dieser Arbeit stehen. In der Dissertation [Hen11] wird der Algorithmus detaillierter behandelt. Dennoch ist es wichtig, die Rendergleichung zu erwähnen, da damit die physikalisch korrekte Beleuchtung simuliert wird und es der aktuelle Stand der Berechnung von globaler Beleuchtung ist.
Zusammenfassend wird festgehalten, dass ein Raytracealgorithmus für die Synthese der Eingabebilder für das Warping verwendet wird, weil die Entfernung von der Kamera zur Geometrie bekannt sein muss. Ebenso liegt die benötigte Länge eines Sekundärstrahls vor.

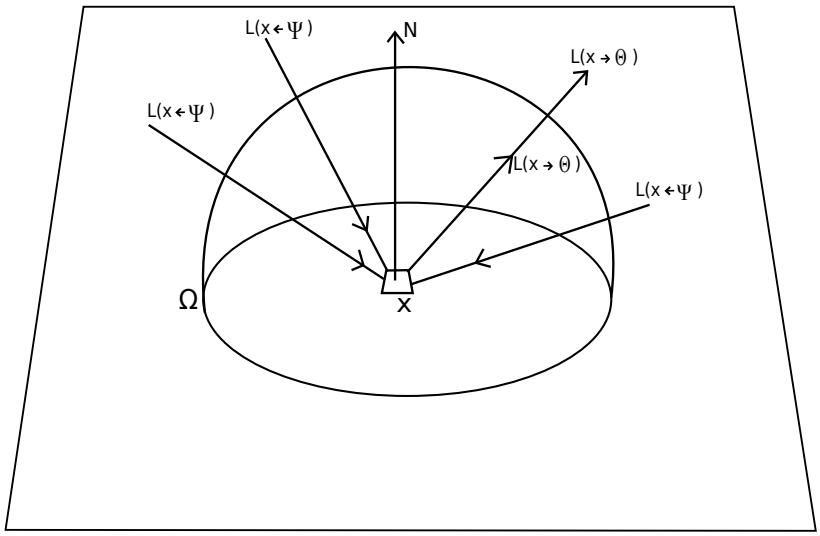


Abbildung 6: Darstellung der Rendergleichung

2.2 Image Warping

Image Warping, kurz Warping, ist ein Bildsynthesealgorithmus, für den einzelne Bilder einer Szene als Eingabe dienen. Der Algorithmus liefert als Ausgabe ein Bild der Szene aus neuem, beliebigem Blickwinkel.
1997 veröffentlichten Leonard McMillan, Gary Bishop und William Mark das Konzept des 3D Warpings [MJMB97]. McMillans Warpingalgorithmus ist auf die Berechnung von diffusen Oberflächen beschränkt, Reflexionen und Brechungen des Lichts sind nicht überzeugend darstellbar. Das Verfahren um auch diese Effekte exakt darstellen zu können, wurde 2014 von G. Lochmann, B. Reinert, T. Ritschel S. Müller und H.-P. Seidel veröffentlicht [LRR$^+$14].
Der Vorteil des Warpings ist, dass das neue Bild mit weniger Rechenaufwand entsteht als das Eingabebild. Daraus ergeben sich unter anderem folgende Nutzungsgebiete:
Durch das Bild mit zweiter, berechneter Perspektive, ist es möglich die Szene ressourcensparend stereo zu rendern, also eine dreidimensionale Sicht zu erlangen. Bei einem Client-Server-System birgt der Algorithmus Vorteile in Hinblick auf die Performanz. Auf der Serverseite finden die rechenaufwändigen Rendervorgänge statt, während auf der Clientseite der Warpingalgorithmus abläuft. Verzögerungen beim Bildempfang kann reduziert werden, da der Client in der Lage ist, Bilder für klein winklige Änderungen der Kamera selbst zu berechnen. Auch in der wachsenden Branche der virtuellen Realität ist das Verfahren von Nutzen. Head-Mounted Displays, *HMDs*, sind Brillen, die Bilder nah am Auge erzeugen. Das Eintauchen in die virtuelle Umgebung, die Immersion, wird gesteigert, wenn der Benutzer nur noch

den Bildschirminhalt und nichts von der Außenwelt sieht. Dabei entsteht bei niedriger Bildfrequenz eine Art „Seekrankheit“, die *Motionsickness*. Um dieser Erscheinung entgegen zu wirken, kann die Bildfrequenz mit dem Warpingalgorithmus stabilisiert werden.

2.2.1 Grundsätzliche Vorgehensweise

Das Image-Warping wird in zwei Schichten aufgeteilt. Diffuse und reflektierende Oberflächen werden dabei getrennt behandelt.
Bei der Aufteilung in Layer wird von zwei verschiedenen Reflexionsarten ausgegangen. Die planare und die diffuse Reflexion. Bei der planaren Reflexion wird von einer glatten, spiegelnden Fläche ausgegangen, bei der diffusen Reflexion von einer Oberfläche, die nicht eben ist und deren Normalen nicht alle in die gleiche Richtung zeigen. Dieses Verhalten ist bei reflektierenden Kugeln bspw. zu beobachten.
Um die Szene und ihren Aufbau repräsentieren und neu berechnen zu können, werden die Tiefen- und Farbinformationen pro Pixel, der Augpunkt, die Ausrichtung der Kamera und der Centerpunkt benötigt [WV12]. Die Tiefen- und Farbinformationen müssen für den diffusen und spekularen Farbanteil getrennt vorliegen, da die Berechnungen für den neuen Blickwinkel für beide Anteile verschieden sind. Die oben genannten Kameraparameter sind Eingaben des Nutzers und für das Eingabebild und für das Ausgabebild bekannt. Die Informationen pro Pixel, Farbe und Tiefe, sind nur für das Eingangsbild bekannt. Diese sollen durch den Algorithmus an eine neue, dreidimensionale Position im Ausgabebild verschoben werden, siehe Abbildung 7

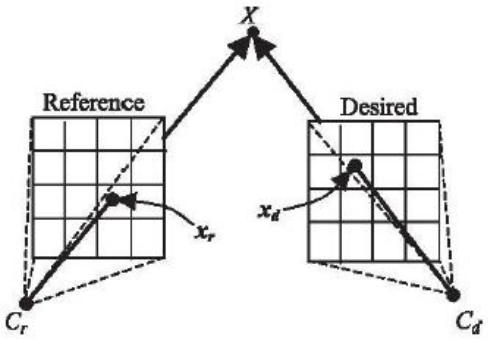


Abbildung 7: Aufbau des Image Warpings

Entnommenaus:[WV12]

2.2.2 Diffuser Anteil

Um das Warping im dreidimensionalen Raum zu verstehen, wird im folgenden Abschnitt das planare Warping [MJ97] erklärt, welches durch Erweiterungen ,wiederum von McMillan, verbessert wird [MJMB97].
Die Szene wird mit Grundlagen des Raytracealgorithmus beschrieben: Es liegen ein Augpunkt und eine Bildebene vor, durch die Strahlen verlaufen. Die Koordinaten der Bildebene werden folgendermaßen als Strahlenvektor dargestellt:

$$\vec{d} = \begin{bmatrix} d_i \\ d_j \\ d_k \end{bmatrix} = \begin{bmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{7}$$

$\vec{d}$ ist der resultierende Strahlenvektor. u und v repräsentieren die Koordinaten der Bildebene, welche als Strahlen dargestellt werden sollen. $\boldsymbol{P}$ beschreibt die Matrix, die das *Mapping* der Koordinaten der Bildebene in das Koordinatensystem der Strahlen vornimmt. Wie im vorherigen Kapitel beschrieben, ist dieses Verfahren gleichzusetzen mit dem eines Raytracers. Zur Übersicht wird die Strahlengleichung eines getroffenen Punktes wiederholt.
Die Indizes der folgenden Gleichungen sind für die Einheitlichkeit dieser Arbeit angepasst. Bei den Gleichung steht der Index d für *desired*, für Werte des zu berechnenden Bildes. Der Index r steht für *reference*, für das Eingangsbild. In den genannten Quellen werden andere Indizes genutzt.

$$\vec{p} = c + t\boldsymbol{P}x$$

$\vec{p}$ ist der Punkt in der Szene. c der Augpunkt, von dem der Strahl ausgeht und t ist ein Skalar, der den Strahl skaliert und den Schnittpunkt mit $\vec{p}$ bestimmt. $\mathbf{P}\vec{x}$ stammt aus Gleichung 7, wobei x aus einer u und einer v Koordinate besteht. Aus dieser Darstellung folgt die Gleichung der Abbildung eines Punkts aus unterschiedlichen Blickwinkeln [MJ97]:

$$\vec{p} = c_r + t_r\boldsymbol{P}_r x_r = c_d + t_d\boldsymbol{P}_d x_d \tag{8}$$

Diese Gleichung hebt hervor, dass ein Punkt $\vec{p}$ von zwei unterschiedlichen Blickwinkeln aus beschrieben werden kann, was in Abbildung 7 bereits visualisiert wird. Stellt man die Gleichung mit dem Ziel um, den Strahl vom Eingangsbild als Strahl des Ausgabebildes darzustellen, erhält man folgende Gleichung [MJ97]:

$$\frac{t_d}{t_d}\mathbf{P}_d x_d = \frac{1}{t_r}(c_r - c_d) + \mathbf{P}_d x_r \tag{9}$$

Der Faktor $\frac{t_d}{t_r}$ auf der linken Seite der Gleichung ist vernachlässigbar, unter der Annahme, dass $\mathbf{P}_r$ und $\mathbf{P}_d$ von Skalierung nicht beeinflusst werden, wodurch $\mathbf{P}_d \frac{t_d}{t_r}$ in sich aufnimmt und unverändert bleibt [MJ97]. Der Skalar der

rechten Seite $\frac{1}{t_d}$ wird durch $\delta(x_r)$ substituiert, die Gleichung wird vereinfacht zu:

$$\mathbf{P}_d x_d = \delta(x_r)(c_r - c_d) + \mathbf{P}_r x_r \tag{10}$$

$\delta(x_r)$ gibt die Struktur der Szene wieder, in der die Korrespondenz eines Punktes aus mehreren Eingangsbildern enthalten ist [WV12]. McMillan nennt sie „verallgemeinerte Ungleichheit“. Sie kann als eine Form des Tiefenwerts verstanden werden [MJMB97]. Das Ziel ist die Position von x_d, danach gilt es die Formel umzustellen [WV12]:

$$x_d = \delta(x_r)\mathbf{P}_d^{-1}(c_r - c_d) + \mathbf{P}_d^{-1}\mathbf{P}_r x_r \tag{11}$$

$\mathbf{P}_d^{-1}$ ist die inverse Matrix von $\mathbf{P}_d$ und x_d ist der gesuchte, neue Punkt.
Für jeden Punkt des Eingabebildes muss $\delta(x_r)$ bekannt sein. Auch die Parameter der Kamera müssen für die Berechnung vorliegen. Daraus wird die Warpinggleichung zu einem linearen Ausdruck vereinfacht [WV12]:

$$\begin{aligned} u_d &= \frac{k11 \cdot u_r + k12 \cdot v_r + k13 + k14 \cdot \delta(u_r, v_r)}{k31 \cdot u_r + k32 \cdot v_r + k33 + k34 \cdot \delta(u_r, v_r)} \\ v_d &= \frac{k21 \cdot u_r + k22 \cdot v_r + k23 + k24 \cdot \delta(u_r, v_r)}{k31 \cdot u_r + k32 \cdot v_r + k33 + k34 \cdot \delta(u_r, v_r)} \end{aligned} \tag{12}$$

Die Zwischenschritte für die Umwandlung sind in McMillans Dissertation nachzulesen [MJ97].
Neu bei Gleichung 12 ist die Abkürzung durch die k's. Sie stehen für die bereits bekannten Kameraparameter der beiden Blickwinkel, welche sich wie folgt zusammensetzen [WV12]:

$$\begin{bmatrix} k11 & k12 & k13 \\ k21 & k22 & k23 \\ k31 & k32 & k33 \end{bmatrix} = \mathbf{P}_d^{-1}\mathbf{P}_r \tag{13}$$

$$\begin{bmatrix} k14 \\ k24 \\ k34 \end{bmatrix} = \mathbf{P}_d^{-1}(C_r - C_d)$$

Damit liegt die in dieser Arbeit verwendete Warpinggleichung vor, mit der Bilder der Szene aus einem neuen Blickwinkel berechnet werden können. Dabei können Rotation und Translation der Kamera verarbeitet werden.
Sobald mehrere Punkte nach dem Warping auf einer Linie liegen, muss entschieden werden, welcher Punkt dem Betrachter angezeigt wird. Um dieses Problem zu lösen, muss keine Distanzberechnung von jedem Punkt zum Betrachter stattfinden. McMillan stellt dafür den Sichtbarkeits Algorithmus vor [MJ97], wodurch ein zusätzlicher Z-Buffer gespart wird. Mit Hilfe von Epipolargeometrie wird eine Traversierungsordnung für die Positionen erstellt, welche gewährleistet, dass die der Kamera nächsten Punkte im Ausgabebild angezeigt werden [WV12]. Epipolargeometrie setzt Bildpunkte in Beziehung,

die aus verschiedenen Winkeln aufgenommen wurden. Dafür wird ein Referenzpunkt ermittelt, der die Warpingreihenfolge bestimmt. Dieser Punkt wird epipolarer Punkt genannt und wird folgendermaßen berechnet [WV12]:

$$\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} = \mathbf{P}_r^{-1}(c_d - c_r) \tag{14}$$

Wie in Abbildung 8 verdeutlicht wird, wird der Augpunkt des neuen Bildes c_d auf die Bildebene des Referenzbildes projiziert.
Anschließend werden die Bildkoordinaten des epipolaren Punktes errechnet:

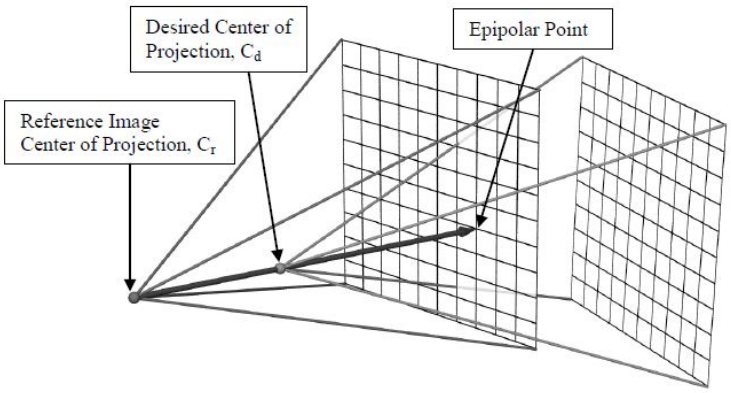


Abbildung 8: Abbildung des epipolaren Punktes auf dem Referenzbild

Entnommenaus:[WV12]

($e = \frac{e_x}{e_z}, \frac{e_y}{e_z}$) und e wird als positiv ($e_z > 0$) oder negativ ($e_z < 0$) eingestuft. e fällt durch die Projektion in einen von neun Bereichen des Referenzbildes, wie Abbildung 9 darstellt.

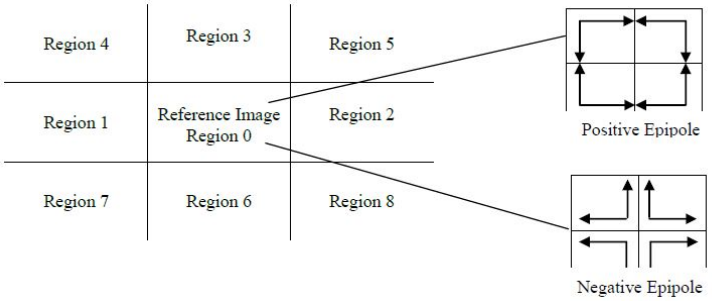


Abbildung 9: Traversierungsreihenfolge im Überblick

Entnommenaus:[WV12]

Je nachdem, ob die Koordinaten von e in dem momentan betrachteten Bereich liegen, wird dieser in vier, zwei oder einen Teilbereich unterteilt

[MJ97]. In vier Bereiche wird die Region aufgeteilt, sobald die Koordinaten von e beide in der Region liegen. Die Aufteilung erfolgt dann horizontal und senkrecht ausgehend von e. Der zweite Fall tritt ein, wenn nur eine Koordinate sich in einer Region befindet. Dann wird diese, wie in Abbildung 10 visualisiert, in zwei Teile unterteilt. Der letzte Fall tritt ein, wenn keine Ko-

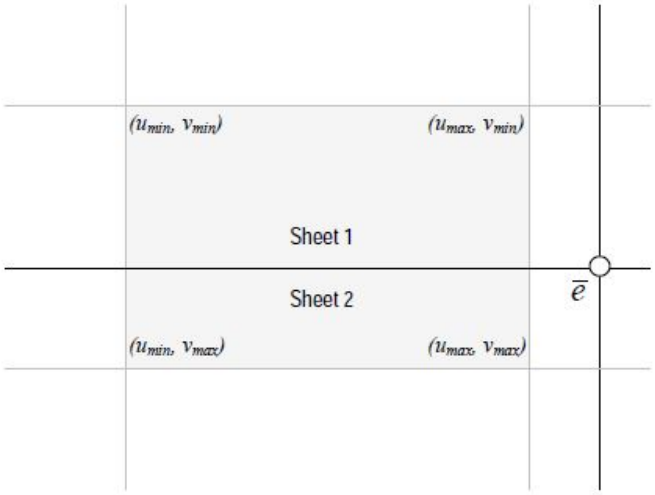


Abbildung 10: Aufteilung einer Region

Entnommenaus:[WV12]

ordinate von e in der Region liegt. Dann wird diese nicht weiter aufgeteilt. Sobald alle Regionen unterteilt sind, wird der positive/negative epipolare Punkt e mit der Unterteilung der Regionen kombiniert und die Reihenfolge der Traversierung wird festgelegt [MJ97]. Wenn e positiv ist, wird vom Rand der Region in Richtung zu e hin geordnet. Falls e negativ ist, wird die Richtung umgekehrt [MJ97]. Dargestellt ist dieses Vorgehen in Abbildung 11.

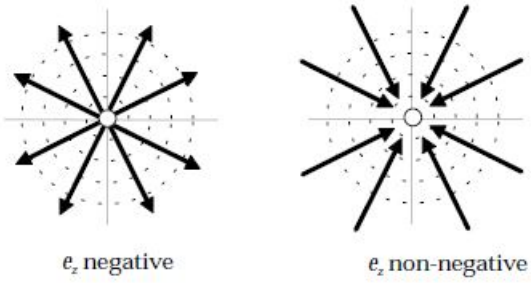


Abbildung 11: Traversierungsreihenfolge ausgehend von e

Entnommenaus:[WV12]

Ist e=0, bedeutet das, dass die neue Sicht nicht korrekt auf das Referenzbild abgebildet werden kann [MJ97]. Der Vektor $c_d - c_r$ ist dann parallel zur Bildebene des Referenzbildes. In diesem Fall wird bei der Warpingreihenfolge von einer ersten Region ausgegangen und die weitere Reihenfolge erschließt sich entlang der Ausrichtung der Vorzeichen von e_x und e_y.
Mit diesem Algorithmus wird erreicht, dass kein weiterer Tiefentest nötig ist und der Betrachter immer das vorderste Pixel sieht.
Aufbauend auf diesem Vorgehen, kann der Warpingalgorithmus verbessert werden. Ein weiterer Effekt, der verhindert werden soll, ist das Auftreten von Löchern im Ausgabebild. Das Verstehen von Geometrie als Punkte im Raum, das beim Warping benutzt wird, wird *Splatting* genannt und fällt in die Kategorie der Rendertechniken. Da keine Relation zwischen einzelnen Punkten besteht, wird jeder Punkt einzeln behandelt. Dabei wird nicht berücksichtigt, ob Formen und Flächen beibehalten werden. Um diese Löcher zu schließen, die besonders auf Flächen auftreten, schlägt [LRR+14] die Kombination von zwei Algorithmen vor. Zuerst werden die Löcher geschlossen, indem ein benachbartes Pixel gewählt wird, das der Kamera am nächsten ist. Dieses Pixel ersetzt dann das fehlende. Hierbei entsteht ein weiteres Problem: Kanten werden ausgebreitet, wodurch Objekte ungewollt wachsen. Um dieses Problem zu lösen, wird die Fixpunktiteration, FPI, vorgeschlagen. Mit der FPI wird die Form der Geometrie iterativ angenähert, wodurch nach dem Warping die Form beibehalten werden kann. Detailliert wird die FPI in dieser Veröffentlichung behandelt: [BMS+12].
Folgende Erweiterung reduziert Artefakte im Ausgabebild, ist für die Umsetzung dieser Arbeit jedoch nicht von Belang: In der Veröffentlichung [MJMB97] wird vorgeschlagen, eine Rekonstruktion des *Meshes* zu nutzen. Dadurch soll die Form der Geometrie beibehalten werden. Detaillierter wird die Funktionsweise in der Veröffentlichung [MJMB97] beschrieben.
Bei der Bildsynthese für einen neuen Blickwinkel mit nur einem Eingangsbild, treten zwangsläufig Artefakte im Ausgabebild auf. Etwa bei Rotation um ein Objekt, liegen weder Farb- noch Tiefeninformationen über die Oberfläche vor, die im Eingangsbild verdeckt ist. Im Ausgabebild tritt dadurch der Fall ein, dass mit undefinierten Pixeln umgegangen werden muss. Schwarz einfärben, wie in Abbildung 12 zusehen, ist eine Darstellung dafür. Sobald mehrere Referenzbilder vorliegen, können durch geschickte Auswahl der Eingangsbilder, viele Informationen über die Szene gesammelt werden [MJMB97]. Daraus ergeben sich weniger undefinierte Pixel, da die Eingangsbilder kombiniert werden können. Auch dieser Ansatz wird in dieser Arbeit nicht umgesetzt und wird deshalb nicht genauer erklärt.

2.2.3 Reflektierender Anteil

Ergänzend zum diffusen Anteil der Farbe, wird nun der reflektierende Anteil für das Warping behandelt. Die folgende Erläuterung basiert auf der Veröf-

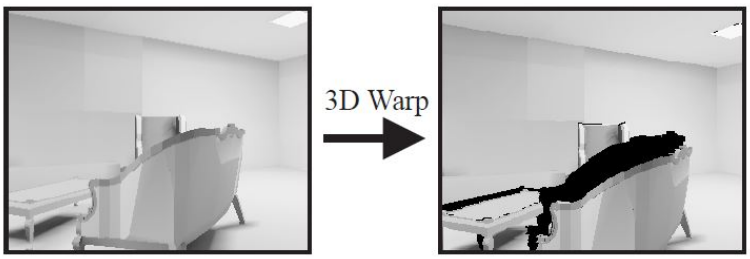


Abbildung 12: Darstellung von undefinierten Stellen nach dem Warping

Entnommenaus:[MMB97]

fentlichung [LRR+14] und deren Implementation.
Bevor wie mit dem Warping von diffusen Oberflächen verfahren werden kann, ist eine weitere Berechnung notwendig. Damit die Reflexion nicht planar, sondern plastisch erscheint, muss die Position der Reflexion vor dem Warping neu ermittelt werden. Hierfür werden die Eingangsposition der diffusen Oberfläche, die Normale an dieser Stelle und die Eingangsposition der Reflexion benötigt. Anhand der Abbildung 13 soll die Bestimmung der neuen Position verdeutlicht werden.

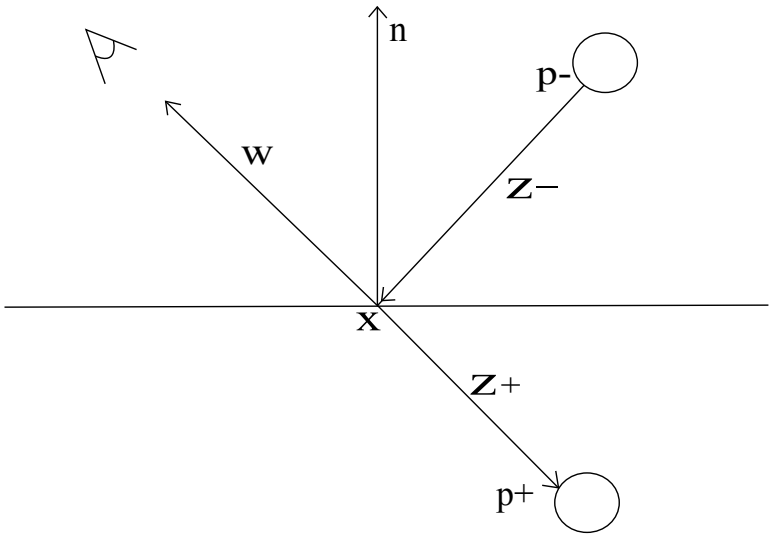


Abbildung 13: Berechnung der neuen Position der Reflexion

Von dem betrachteten Punkt x, der die diffuse Weltposition enthält, wird die Position der dort sichtbaren Reflexion p- benötigt, um den Vektor z- zu ermitteln. Dieser wird anschließend mit Hilfe der Normale n an der diffusen Position reflektiert, Vektor w entsteht. Der plastische Effekt wird erreicht, indem der reflektierte Vektor w von der diffusen Position x subtrahiert und mit der Länge von z- multipliziert wird. z+ ist damit berechnet. Durch die-

se Berechnung erhält die Spiegelung die korrekte, neue Position z+. Diese ermittelte Position wird nun wie eine diffuse Oberfläche behandelt und als Eingabe für das eben beschriebene Warping verwendet.
Das Warping ist damit vollständig. Im folgenden Kapitel wird die Zusammenführung der Ergebnisse erläutert.

2.2.4 Blending und dessen Probleme

Dieses Kapitel basiert auf der Veröffentlichung [LRR+14] und deren Implementation.
Durch die Aufteilung in mehrere Layer, entsteht die Problematik, die Schichten wieder zusammen führen zu müssen. Um diese Layer wieder zusammen zu bringen, wird eine Vermischung der Bilder vorgenommen, das *Blending*. Dabei muss entschieden werden, welche Art der Reflexion an der gerade betrachteten Position korrekt ist und dem Betrachter angezeigt werden soll.
Als Eingabe für das Blending werden beide Reflexionsarten benutzt, die es dann zu interpolieren gilt. Der Interpolationsfaktor wird von einer Funktion ermittelt, die die Güte der Eingabeposition bestimmt, der Qualitätsfunktion. Die Interpolation setzt sich wie folgt zusammen:

$$\begin{aligned} p_0 &= \lambda f_d + (1-\lambda) f_s \\ \lambda &= \tfrac{1-\Delta_d}{\Delta_d+\Delta_s} \\ \Delta_d &= \Delta(\vec{\omega}, W_d(l + f_d(l))) \\ \Delta_s &= \Delta(\vec{\omega}, W_s(l + f_s(l))) \end{aligned} \tag{15}$$

p_0 entsteht durch normale Interpolation zwischen den beiden Eingabepositionen f_d und f_s, wobei die Indizes für diffuse und spekulare Reflexion stehen. Δ_d entspricht dem Wert, der die Qualitätsfunktion liefert. Diese wertet die Abweichung zwischen einem Referenzvektor $\vec{\omega}$ und der Weltposition der diffusen Reflexion aus. Der Referenzvektor wird aus der diffusen Weltposition, deren Normale und dem neuen Augpunkt berechnet und ist der rekonstruierte Reflexionsvektor. Je kleiner der Winkel zwischen Reflexionsvektor und dem Vektor zur Weltposition der gespiegelten Position ist, desto korrektere Ergebnisse werden erreicht. Δ_s steht analog für die Güte der spekularen Eingabeposition.
Der Interpolationsfaktor λ ergibt sich aus zurückgegebenen Werten der Qualitätsfunktion. Diese interpolierte Position wird anschließend als Eingabe für eine weitere Funktion benutzt, die die Position auf Güte testet und gegebenenfalls im Umkreis der eingegebenen Position nach Ergebnissen sucht, die dem Referenzvektor näher sind und deshalb die optische Qualität steigern. Die Interpolation darf nicht auf primitive Weise geschehen, da sonst nicht gewährleistet ist, dass physikalisch korrekte Spiegelungen berechnet werden. Es muss eine Funktion über die physikalische und optische Güte entscheiden,

welche Art der Spiegelung an der entsprechenden Stelle gewählt werden soll.

3 Konzept und Implementation

In diesem Kapitel wird die Umsetzung der Theorie behandelt. Es wird anfangs darauf eingegangen, welche der vorangegangenen Grundlagen umgesetzt werden, um das Ziel dieser Arbeit zu erreichen. Außerdem wird begründet, warum nicht alles implementiert wird, bzw. warum dies nicht nötig ist. Abgeschlossen wird dieses Kapitel mit der Dokumentation der Implementation.
Es wird ein Raytracer zur Erstellung der Eingabetexturen gewählt, da mit diesem Algorithmus die Länge von Primär- und Sekundärstrahlen berechnet werden kann. Dabei ist es von Vorteil, dass die Grafikkarte für die Berechnung benutzt wird, da jeder ausgehende Strahl beim Raytracing unabhängig berechnet werden kann. Durch diese Parallelisierung wird Rechenlast eingespart und Echtzeit bleibt erhalten. Implementation und Tests sind mit einer GTX 770 Grafikkarte von Nvidia, einer CPU mit 4 Kernen à 3,4 GHz und 8 GB RAM durchgeführt worden.

3.1 Konzept

Aus den in vorherigen Kapiteln vorgestellten Gleichungen ergeben sich folgende benötigte Texturen als Eingabe für das Warping: die Position, Farbe und Normale der diffusen Oberfläche und die reflektierte Position. Die Tiefe ist in der Position der einzelnen Layer enthalten.
In dieser Arbeit wird der gebrochene Anteil des Lichts nicht beachtet. Das Warping der reflektierten Farbe genügt zur Darstellung des Reflective-Warping Verfahrens.
Damit das Verfahren echtzeitfähig bleibt und weil das Raytracing nicht im Mittelpunkt der Arbeit steht, wird der Raytracealgorithmus simpel und rudimentär gehalten. Es wird zum einen nur ein Strahl pro Pixel verfolgt und auch die Reflexion verfolgt nur einen Strahl. Zum anderen wird Phong Shading gewählt, um interpolierte Normalen zu erhalten. Das Phong-Beleuchtungsmodell soll jedoch nicht vollständig implementiert werden, da Glanzlichter für diese Arbeit nicht relevant sind. Es wird folglich keine globale Beleuchtung implementiert. Für das Warping ist diese auch nicht notwendig. Außerdem werden Schatten nicht berechnet. Sie sind für das Verfahren nicht relevant und wären rechenintensiv.
Wie bereits erwähnt, beschränkt sich das Warping auf den diffusen und den reflektierten Anteil des Lichts. Diese werden beide getrennt behandelt, da unterschiedliche Berechnungen nötig sind. Als Eingabe für das diffuse Warping wird die Weltposition der diffusen Oberflächen, die Normale an der Stelle und die Parameter des neuen Blickwinkels benötigt. Ausgabe nach diesem

Warping ist die diffuse Position und die Normale aus dem neuen Blickwinkel. Außerdem wird für das folgende Blending die uv-Koordinate der Ausgangsposition in der neuen Position nach dem Warping gespeichert. Im Abschnitt der Implementation wird die Reduzierung der entstehenden Löcher behandelt.
Um die Reflexion zu berechnen, dienen die diffuse Position, die Normale, die Position der Reflexion und die neuen Kameraparameter als Eingabe. Die Ausgabe umfasst die uv-Koordinate der alten Reflexionsposition an der neu ermittelten Position, analog zu der diffusen uv-Koordinate.
Für das Blending der beiden Reflexionsarten liegen danach die Eingabetexturen vor: Die Farbe und Position der spiegelnden Oberfläche aus dem Raytracepass, die neue Positionen der diffusen Oberfläche, deren Normale und die uv-Koordinaten aus den beiden Warpingpasses. Das Ergebnis des Blendingpasses soll eine Textur sein, bei der für jede Oberfläche die korrekte Reflexionsart gewählt und angezeigt wird. Das ursprüngliche Bild der Szene, das bei dem Raytracealgorithmus entstanden ist, ist nach dem Blending bzw. Compositing vollständig aus einem beliebigen anderen Blickwinkel betrachtbar. Dabei wird die diffuse Farbe der Objekte, ebenso wie die reflektierte, korrekt ausgegeben. Die Optimierung des Blendingpasses soll das Verfahren performanter machen und optisch aufwerten. Ob und wie die Optimierung funktioniert, wird im späteren Verlauf der Arbeit untersucht.
In Abbildung 14 soll der Aufbau und Ablauf der Renderpasses verdeutlicht werden.

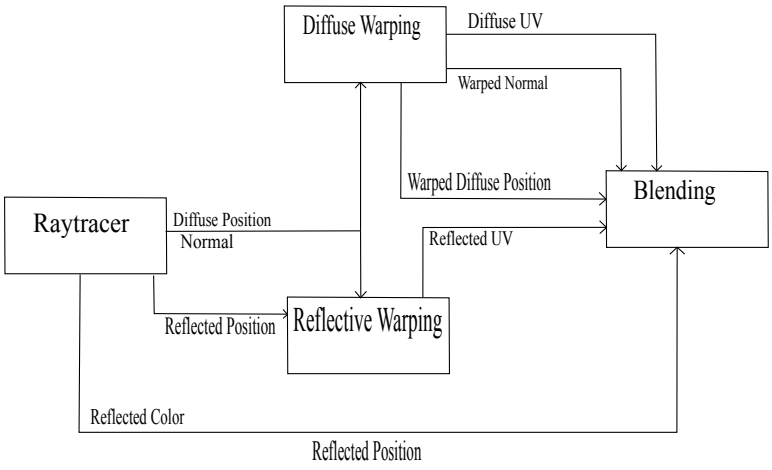


Abbildung 14: Übersicht des Konzepts

3.2 Implementation

3.2.1 Genutzte Mittel

OpenGL: OpenGL ist eine freie Grafikbibliothek, mit der dreidimensionale Szenen programmiert werden.

Framework: Die Implementation basiert auf einem bereitgestellten OpenGL Framework.

GLSL: Um auf dem Grafikprozessor programmieren zu können, wird die OpenGL Shading Language, GLSL, genutzt. In dieser Arbeit die Version 4.3.

C++: Version elf

GLFW: Für Tastatur- und Mauseingaben, sowie die Fenstererstellung, wird die ebenfalls freie Bibliothek GLFW verwendet, die mit OpenGL Inhalten arbeitet.

Glew: Eine C/C++ Bibliothek, die die Verwendung von OpenGL-Erweiterungen unterstützt.

GLM: Eine C++ Mathematikbibliothek für Grafikprogramme.

3.2.2 Implementation der Renderpasses

Die Erstellung der Szene erfolgt über mehrere Wege. Der Hintergrund ist eine Textur, die mit Hilfe von *Environment Mapping* abgebildet wird. Die Primärstrahlen tasten diese Textur ab:

```
1 vec3 background =
2 vec3(texture(environmentTexture,
3 vec2(atan(rd.x, rd.z)/ 6.2832 + 0.5, acos(rd.y) /3.1416)));
```

Das Ergebnis ist eine Farbe, gespeichert in einem vec3. Dafür erfolgt ein Texturaufruf an der Stelle $vec2(atan(rd.x, rd.z)/6.2832+0.5, acos(rd.y)/3.1416)$, wobei rd für den Vektor steht, der die Richtung des Sehstrahls angibt. Der Hintergrund wird als diffuse Farbe, aber nicht als getroffene Geometrie gehandhabt. Dadurch wird der Hintergrund absichtlich nicht als reflektierende Fläche im Warpingalgorithmus angesehen.
Die Kugeln in der Szene werden als Vektoren in einem vec4 Vektor übergeben:

```
1 vector<vec4> sphereVec;
2 sphereVec.push_back(glm::vec4(0.0, 0.0, 0.0, 0.5));
3 sphereVec.push_back(glm::vec4(-0.5, 0.5, 1.0, 0.3));
```

Im Shader wird dieser Vektor als Uniformvariable entgegen genommen:

```
1 uniform vec4 sphereVec[2];
```

Da nur zwei Kugeln für den Aufbau der Szene benutzt werden, ist diese Methode effizient genug. Sobald jedoch mehr Daten für die Objektbeschreibung benötigt werden, taucht ein technischer, limitierender Faktor seitens der Grafikkarte auf, der nicht mehr als zwölf Vektoren zulässt. Für komplexere Objekte, die bspw. mit Dreiecken modelliert werden, genügt diese mögliche Anzahl von Vektoren nicht. Um die spiegelnden Flächen, die aus Dreiecken bestehen, darzustellen, wird ein minimaler Objektlader [1] benutzt. Der Objektlader übergibt die Vertices und deren Normalen an einen Shaderstoragebuffer [2]. Mit diesem ist es möglich, die Limitierung der Vektorenanzahl zu umgehen. Shaderstoragebuffer sind den Uniform Buffer Objekten sehr ähnlich und sind in der Lage Daten innerhalb GLSL lesend und schreibend zu verarbeiten. Die bereitgestellte Kapazität ist nützlich für das Laden von Objekten. Die Daten der Dreiecke werden im Shader als Interface Block bereitgestellt:

```
1 layout(std430, binding=11) buffer meshData{
2         vec4 pos[300];
3 } myMesh;
4
5 layout(std430, binding=12) buffer meshNormal{
6         vec4 posNorm[300];
7 } myNormals;
```

Der Binding-Befehl ist in diesem Fall notwendig, um den ShaderstorageBuffern explizit einen Speicherort zuzuweisen. std430 beschreibt das Speicherlayout, also wie die Daten gepeichert werden. Die Spezifikation ist der OpenGL Seite zu entnehmen[3].
Materialeigenschaften werden in dieser Arbeit nicht berücksichtigt, da sie nicht relevant für das Warpingverfahren sind. Lediglich unterschiedliche Farben werden der Geometrie zugewiesen.
Die Lichtquelle besteht aus einem Vektor, der fest in der Szene definiert ist. Mit dieser vorbereiteten Szene kann nun die Bildsynthese mit dem Raytracealgorithmus stattfinden.

Der folgende Codeausschnitt berechnet die Kameraparameter. Augpunkt, Bildebene und die Primärstrahlen setzen sich folgendermaßen zusammen:

```
1 vec3 currentPos = (invView * vec4(0,0,0,1)).xyz;
2 vec2 fragCoord = vec2(gl_FragCoord.xy/ resolution.xy)* 2 -
      1;
3
```

[1] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/
[2] https://www.opengl.org/registry/doc/glspec44.core.pdf
[3] https://www.opengl.org/registry/doc/glspec44.core.pdf

```
4 vec3 currentDir = normalize(inverse(mat3(projection)* mat3(
      view))* vec3(fragCoord,1));
```

CurrentPos ist der festgelegte Centerpunkt, hier ist es der Ursprung in Weltkoordinaten. Mit *fragCoord* wird die Bildebene beschrieben, die die gewünschte Auflösung berücksichtigt und durch die Skalierung $[..] * 2 - 1$ das gesamte Ausgabebild einnimmt. In dieser Arbeit wurde mit einer Auflösung von 1280*720 Bildpunkten gearbeitet. *currentDir* ermittelt anschließend den aktuellen Strahl, der in die Szene geschickt wird.
Die Schnittpunktberechnung mit Kugel[4] und Dreieck[5] sollen nicht näher erklärt werden, da beide Funktionen aus fremden Quellen entnommen und nicht relevant für diese Arbeit sind.
Mit den Parametern des Strahls wird als nächstes in einer Schleife jede Geometrie auf einen Schnittpunkt überprüft. Sobald Geometrie getroffen wird, wird die Farbe berechnet und die entsprechende Position und Normale bestimmt. An diesem Punkt des Algorithmus liegt bereits die Textur der diffusen Farbe vor, siehe Abbildung 15.

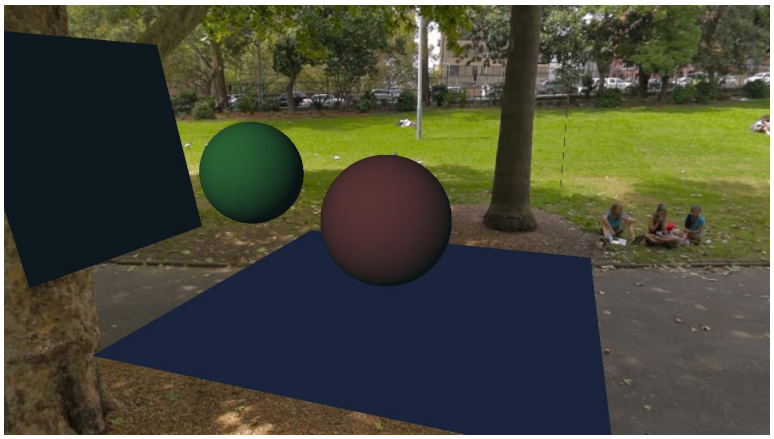


Abbildung 15: Diffuse Farbe

Ebenso liegt die Position der diffusen Oberfläche vor (Abb. 16). Hier wird die Position im Weltkoordinatensystem über die Farbkanäle kodiert: Rot entlang der x-Achse, grün entlang der y-Achse und blau entlang der z-Achse.

Auch die Normale an der getroffenen Geometrie wird als Textur gespeichert, was Abbildung 17 darstellt.

Die Normale beschreibt die Ausrichtung der Oberfläche, dabei ist für diese Arbeit wichtig, dass die Normalen für jedes Pixel individuell berechnet werden. Für das folgende Warping müssen Oberflächen vorliegen, bei denen

[4] https://www.shadertoy.com/view/ldS3DW
[5] http://undernones.blogspot.de/2010/12/gpu-ray-tracing-with-glsl.html

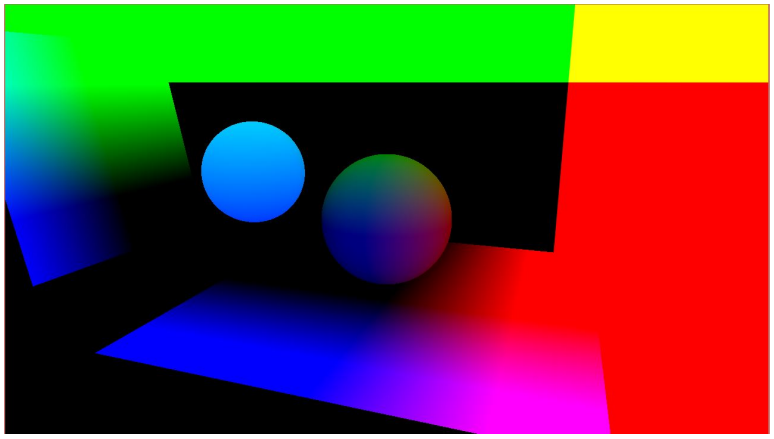


Abbildung 16: Position von diffuser Oberfläche

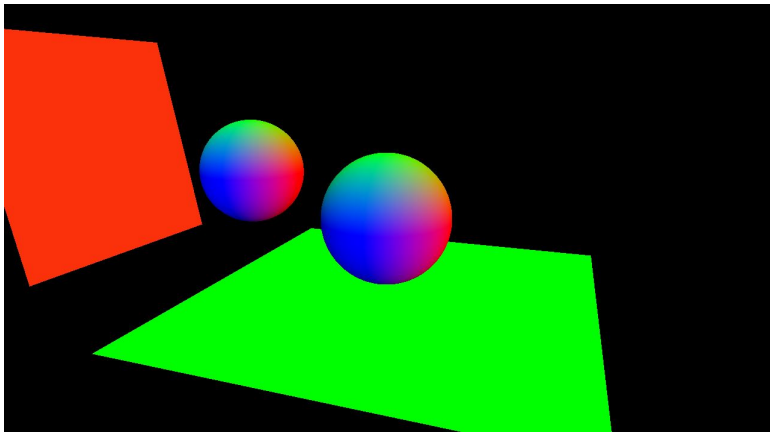


Abbildung 17: Normale

die Normalen variieren, weshalb sie bei dem Raytracing interpoliert werden. Erkennbar sind die Farbverläufe bei den Kugeln. Mit uneinheitlichen Normalen für eine Oberfläche wird die diffuse Reflexion getestet. Einheitliche Normalen, wie sie auf den Flächen zu erkennen sind, testen die gerichtete Reflexion.

Trifft der Strahl keine Geometrie, wird als Farbe der Hintergrund gesetzt. Die Tiefe und damit die Position wird als algorithmus-intern Unendlich behandelt. Durch den großen Wert der Tiefe im Unendlichen entstehen die gesättigten Farben in der Positionstextur. Bei der Textur der Normalen hingegen, wird nur ein Wert gesetzt, falls tatsächlich Geometrie getroffen wird, andernfalls bleibt das Pixel schwarz.
An diesem Punkt des Raytracings folgt anschließend die Trennung zwischen

diffuser und reflektierter Farbe.
Nachdem die bereits genannten Werte berechnet sind, werden ausgehend von diesen die Reflexionsstrahlen berechnet und in die Szene geschickt. Dieser Vorgang kann für eine beliebige Indirektionsanzahl fortgeführt werden, dafür ist die Aufteilung in weitere Layer jedoch notwendig.
Die Textur der reflektierten Farbe ist in Abbildung 18 zu sehen.

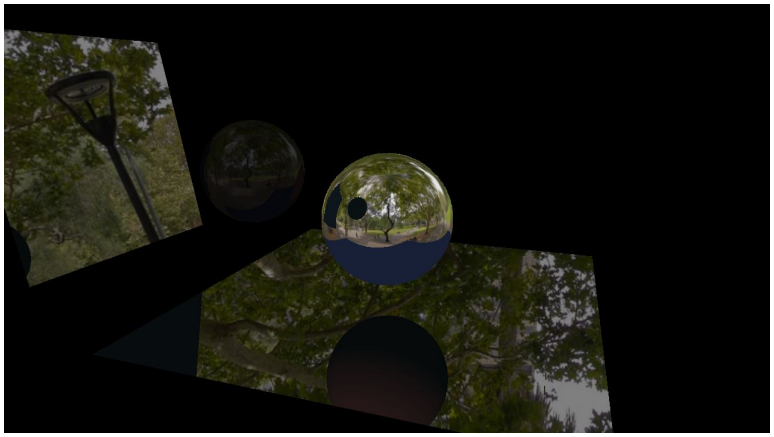


Abbildung 18: Reflektierte Farbe

Wie zu sehen ist, wird die Umgebung auf der Oberfläche der Geometrie gespiegelt. Dass nur eine Indirektion vorliegt ist erkennbar, da in der Spiegelung keine weitere Spiegelung zu sehen ist.
Die Position der reflektierten Geometrie wird ebenfalls kodiert, dargestellt in Abbildung 19.

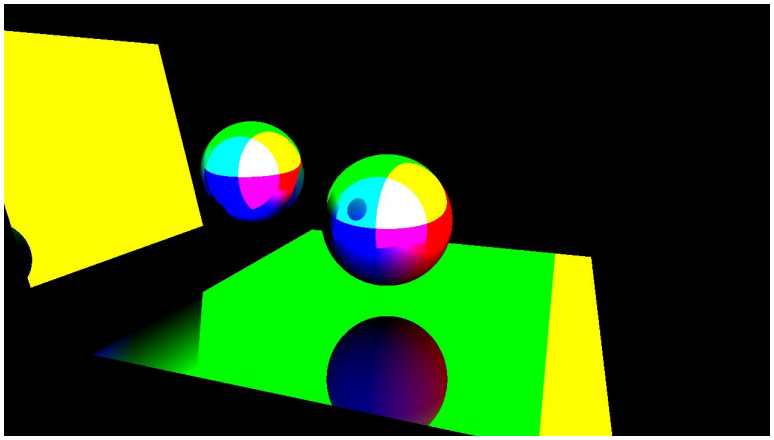


Abbildung 19: Reflektierte Position

An der betrachteten Stelle der Textur ist die Position der Reflexion als Farbwert kodiert. Die gesättigten Farben stehen in diesem Fall für die Reflexion ins Unendliche. Wo schwarz steht, gibt es keine Reflexion.
Aus dem Compositing der Ausgabetexturen vom Raytracer geht ein Bild hervor, dass alle Farbanteile vereint (Abb. 20).

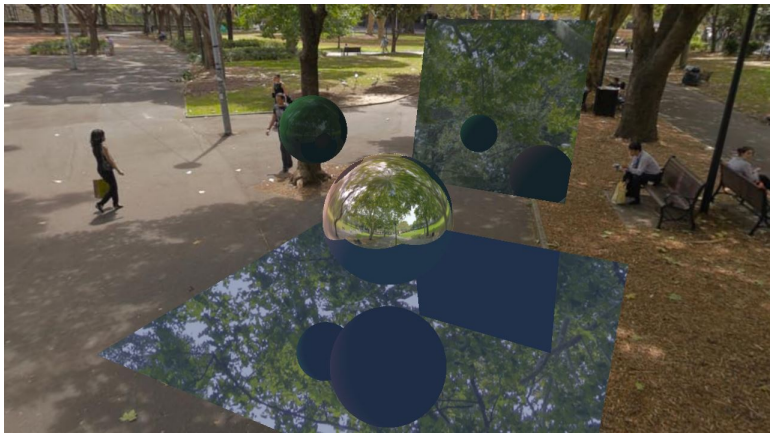


Abbildung 20: Ergebnis des Raytracers mit allen Farbanteilen

Das Ziel dieser Arbeit ist es, dieses Bild nach dem Warpingalgorithmus aus einem anderen Blickwinkel betrachten zu können.
Die vorgestellten Texturen dienen als Eingabe für die zwei Warpingalgorithmen, die im Folgenden behandelt werden.

Für das Rendering des Warpings ist eine grundlegende Änderungen auf OpenGL-Seite notwendig. Statt wie beim Raytracing den *Triangle Strip Modus* bei dem *Vertex Array Object* zu nutzen, werden beim Warping einzelne Punkte gerendert. Dabei wird jedes Pixel einem Punkt zugeordnet. Die Szene wird also als Ansammlung von Punkten verstanden. Da die Punkte in keiner Relation zueinander stehen und keine Verbindungen offensichtlich sind, entstehen, wie im vorherigen Kapitel genannt, nach dem Warping Löcher auf der Geometrie. Diesem Verhalten kann mit dem Befehl *glEnable(GL_POINT_SMOOTH);* entgegen gewirkt werden, indem OpenGL-internes *Antialiasing* aktiviert wird.
Außerdem wird ein Server-Client-System simuliert, indem Latenz absichtlich herbeigeführt wird. Während das Warping mit der aktuellen, „neuen“ Viewmatrix berechnet wird, wird das Raytracing mit einer „alten“ Viewmatrix ausgeführt. Dazu wird die aktuelle Viewmatrix gespeichert und nach beliebig langer Verzögerung als Eingabe für das Raytracing genutzt. Sobald keine Änderung der Kamera stattfindet, sind beide Matrizen identisch und das selbe Bild entsteht bei Raytracing und Warping.

Zuerst wird das Warping von diffuser Position und diffuser Reflexion behandelt. Die Positionstextur der diffusen Oberfläche (Abb. 16) und die Parameter für den neuen Blickwinkel dienen als Eingabe für den Shader:

```
1  #version 430
2
3  in vec2 pos;
4
5  uniform mat4            view;
6  uniform mat4            projection;
7  uniform sampler2D       diffPositionTexture;
8
9  out vec2 passPosition;
10
11 void main() {
12         passPosition = pos;
13         gl_Position = projection * view * texture(
               diffPositionTexture, pos) * vec4(1,1,0.999,1);
14 }
```

Die Werte, die aus der diffPositionTexture gelesen werden, sind bereits in Weltkoordinaten. Aus diesem Grund muss keine Rückprojektion aus dem alten Blickwinkel erfolgen. Die eingelesenen Werte werden folglich nur noch in das Koordinatensystem des neuen Blickwinkels projiziert. Dies geschieht mit der Multiplikation mit der neuen View- und der Pojectionmatrix. Die Multiplikation mit vec4(1, 1, 0.999, 1) stellt sicher, dass auch der Hintergrund bei dem Warpingvorgang ausgegeben wird. Der zweiwertige Vektor pos speichert die Position des Eingabepixels.
Damit steht die Position nach dem Warping fest und alte Werte könne an die neue Position übertragen werden. Mit diesem Shader werden die diffuse Position (Abb. 21) und die Normale (Abb. 22) neu berechnet,sowie die alten uv-Koordinaten gespeichert (Abb. 23). Die Textur der uv-Koordinaten ist für das Blending wichtig. Darauf wird später eingegangen.
Dass die Texturen aus dem Warping entstehen, ist daran erkennbar, dass schwarze Ränder neben der Geometrie auftreten. Das sind undefinierte Stellen, an denen Informationen über Tiefe und Farbe fehlen.

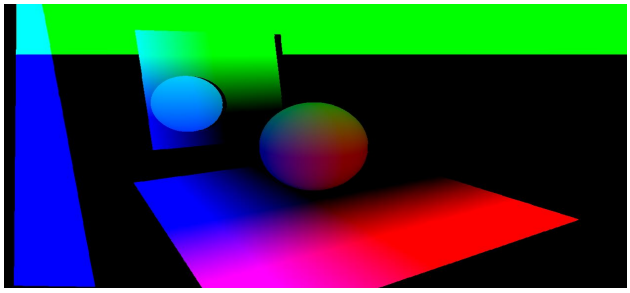


Abbildung 21: Diffuse Position nach dem Warping

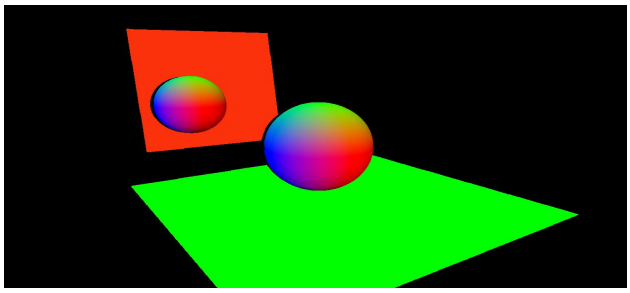


Abbildung 22: Normale nach dem Warping

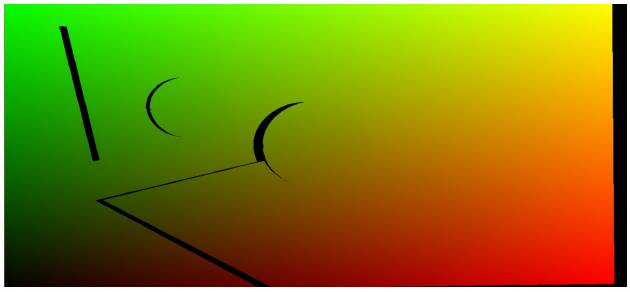


Abbildung 23: uv-Koordinaten der diffusen Position nach dem Warping

Für das Warping der Oberfläche mit gerichteter Reflexion, wird die Position der Reflexion (Abb. 19), die Position der diffusen Oberfläche (Abb. 16), die Normale (Abb. 17) und die neuen Kameraparameter benötigt. Der Shader für das Warping der Reflexion ist wie folgt:

```
1  #version 430
2
3  in vec2 pos;
4
5  uniform sampler2D reflectionPositionTexture;
6  uniform sampler2D diffusePositionTexture;
7  uniform sampler2D normalTexture;
8  uniform mat4 view;
9  uniform mat4 projection;
10 uniform vec2 resolution;
11
12 out vec2 passPosition;
13
14 void main() {
15 passPosition = pos;
16
17 vec3 wDifPos = texture(diffusePositionTexture, pos).xyz;
18
19 vec3 wNorm = texture(normalTexture, pos).xyz;
20
21 vec3 wRefPos = texture(reflectionPositionTexture, pos).xyz;
22
23 vec4 wPos = vec4(wDifPos - normalize(reflect(wDifPos -
       wRefPos,normalize(wNorm))) * distance(wDifPos, wRefPos),
        1);
24
25 gl_Position = (projection * view * wPos)* vec4(1,1,0.999,1);
26 }
```

Bevor das Warping wie für diffuse Oberflächen stattfinden kann (Zeile 25), muss die Position der Reflexion neu berechnet werden. Das passiert hauptsächlich in Zeile 23.
Wie bereits im vorherigen Kapitel beschrieben, wird durch diese Berechnung Plastizität erreicht. Mit der Position der diffusen Oberfläche und der der Reflexion wird mit Hilfe der Normalen der Vektor errechnet, der sich auf der diffusen Oberfläche befindet und zur Kamera zeigt. Dieser wird dann entlang seiner Position in seine umgekehrte Richtung verschoben. Mit der Distanz zwischen diffuser Oberfläche und der Position der Reflexion wird er anschließend skaliert um die korrekte Länge wieder herzustellen. Dadurch erscheinen Reflexion tiefer im Raum. An die neu berechnete Position werden anschließend die die uv-Koordinaten der alten Position geschrieben (Abb. 24).

Abbildung 24: uv-Koordinaten der reflektierten Position nach dem Warping

Die diffuse und die gerichtete Reflexion sind nun vollständig berechnet. Der nächste Schritt behandelt das Blending der resultierenden Texturen. Dabei muss entschieden werden, welche Art der Reflexion an der aktuellen Stelle die richtige ist.
Für das Blending werden die Position von der diffusen Oberfläche nach dem Warping (Abb. 21), die Texturen der beiden uv-Koordinaten (Abb. 23 und Abb. 24) und die Normale nach dem Warping (Abb. 22) benötigt. Außerdem die Farbe und Position der Reflexion aus dem Raytracepass (Abb. 19 und Abb. 18).
Wichtig für das Blending sind die beiden Texturen der uv-Koordinaten. In denen ist die Ausgangsposition vor dem jeweiligen Warping gespeichert. Das bedeutet, dass zwei verschiedene Positionen von Reflexionen vorliegen. Der Blendingalgorithmus entscheidet, welche Position und damit welche Art der Reflexion für den betrachteten Punkt richtig ist. Dafür sind mehrere Schritte notwendig.
Zu Anfang wird ein Referenzvektor ermittelt, der später zum Vergleich benötigt wird. Dieser Vektor reproduziert den reflektierten Vektor auf der diffusen Oberfläche nach dem Warping. Folgendermaßen wird dieser Vektor berechnet:

```
1 vec3 eyePosition = (inverse(view) * vec4(0,0,0,1)).xyz;
2
3 vec2 uv = gl_FragCoord.xy / resolution;
4
5 vec3 warpedDiffusePosition = texture(
      warpedDiffusePositionTexture, uv).xyz;
6
7 vec4 warpedNormal = texture(warpedNormalTexture,uv);
8
```

```
9 vec3 reflectionVector = normalize(reflect(
      warpedDiffusePosition - eyePosition, normalize(
      warpedNormal.xyz)));
```

Mit eyePosition wird der Augpunkt des neuen Blickwinkels ermittelt. Anschließend wird der Vektor von Augposition zur diffusen Oberfläche benötigt, um dort mit Hilfe der Normalen die reflect-Funktion von OpenGL zu benutzen und den Referenzvektor zu erlangen.
Im nächsten Schritt wird die Qualitätsfunktion für beide uv-Koordinaten aufgerufen. Die Qualitätsfunktion gibt jeweils einen float zurück, der für die Güte des erhaltenen Testvektors steht.

```
1 float reflectionQuality(vec2 coord) {
2 vec3 r = normalize(vec4(reflectionVector,1).xyz);
3
4 vec3 testReflectionPosition = texture(
      oldReflectionPositionTexture, coord).xyz;
5
6 vec3 testReflectionVector = normalize(testReflectionPosition
      - warpedDiffusePosition);
7
8 return acos(clamp(dot(r, testReflectionVector), -1, 1));
9 }
```

In testReflectionPosition steht die Position der Reflexion. coord nimmt im Algorithmus die reflektierte Position von diffuser und gerichteter Reflexion ein. Daraufhin wird der zu testende Vektor erstellt, der zwischen diffuser Oberfläche und der Position der Reflexion liegt. Die Rückgabe der Funktion ergibt sich aus dem Winkel zwischen Referenzvektor und dem zu testenden Vektor. Die Ergebnisse der aufgerufenen Qualitätsfunktionen ergeben dann den Interpolationsfaktor. Es wird also zwischen beiden möglichen Reflexionspositionen interpoliert:

```
1 vec2 initialGuess = (uvRef * g2Q + uvDiff * g1Q) / (g1Q +
      g2Q);
```

uvRef und uvDiff stehen für die Koordinaten der Position von der Reflexion. Diese werden mit den Ergebnissen der Qualitätsfunktion gewichtet und anschließend mit der Summe der Ergebnisse normiert. Das Ergebnis der Interpolation wird wiederum als Eingabe für eine Methode benutzt, die das Ergebnis verbessert. Die gradientDescent-Methode:

```
1 vec2 gradientDescent(vec2 initialGuess) {
2     vec2 gradient;
3     float g00;
4     float g01;
5         int maxGDSteps = gradientDescentSteps;
6         float newQuality;
7         // pixel vs. subpixel
```

```
8           vec2 eps = vec2(1, 1) / resolution;
9           float bestQuality = 999.0;
10          float l = 0.00001;
11          float ensuredQualityThreshold = 0.03;
12          bool undefined = false;
13
14          vec2 outputCoord = initialGuess;
15
16      for (int i = 0; i < maxGDSteps; i++) {
17          newQuality = reflectionQuality(outputCoord);
18
19          g00 = reflectionQuality(outputCoord + vec2(eps.x, 0)
                );
20          g01 = reflectionQuality(outputCoord + vec2(0, eps.y)
                );
21
22          gradient = l * vec2(g00 - newQuality, g01 -
                newQuality) / (eps);
23          outputCoord -= gradient;
24
25          if (newQuality < bestQuality) {
26              bestQuality = newQuality;
27          }
28      }
29      if (bestQuality > ensuredQualityThreshold) {
30          undefined = true;
31          discard;
32          }
33      return outputCoord;
34  }
```

Diese Methode, mit Eingabe der interpolierten Position (initialGuess), sucht in der Umgebung dieser Position nach einer Position, die einen kleineren Winkel zwischen Referenzvektor und reflektierter Position liefert. Dadurch wird das Ergebnis verbessert. Da die gradientDescent-Methode flexibel nach einem besseren Ergebnis sucht, ist bei der vorherigen Interpolation nur gefordert, dass eine ungefähre Position geliefert wird. Je genauer jedoch dieser *initialGuess* ist, desto korrektere und artefaktärmere Ergebnisse können erzielt werden.
Wichtig ist die For-Schleife der Methode. In Zeile 16 und 17 wird in der Nachbarschaft von der Eingabeposition mit der Qualitätsfunktion die Güte dieser Nachbarschaftspositionen berechnet. Die beiden erhaltenen Güte-Indizien werden anschließend dazu benutzt, in der jeweiligen Richtung weiter nach der Optimierung zu streben. Der Integer maxGDSSteps gibt an, wie oft dieser Optimierungsschritt geschieht. Je größer er ist, in desto größerer Nachbarschaft kann gesucht werden. Zum Schluss wird überprüft, ob eine gewünschte Qualität erreicht wurde und der Wert wird entsprechend zu-

rückgegeben. Kann eine geforderte Qualität, also ein kleinerer Winkel, nicht gefunden werden, wird mit dem discard-Befehl die Beschreibung des Pixels verhindert, wodurch bei der Ausgabe sichergestellt wird, dass eine geforderte Qualität vorliegt. Dieses Verhalten stellt also die Güte des Algorithmus dar. Abbildung 25 präsentiert das Ergebnis des Warpings nach dem Blending. Zwischen den Reflexionsarten wird deutlich und effektiv getrennt. Auf den planaren Flächen erscheinen plastische Spiegelungen. Auf der Kugel wird auf der oberen Hälfte die Umgebung reflektiert und auf der unteren Hälfte der untere Spiegel. Dass dazwischen keine Lücke entsteht, bedeutet, dass auch die diffuse Reflexion an der richtigen Stelle ist.

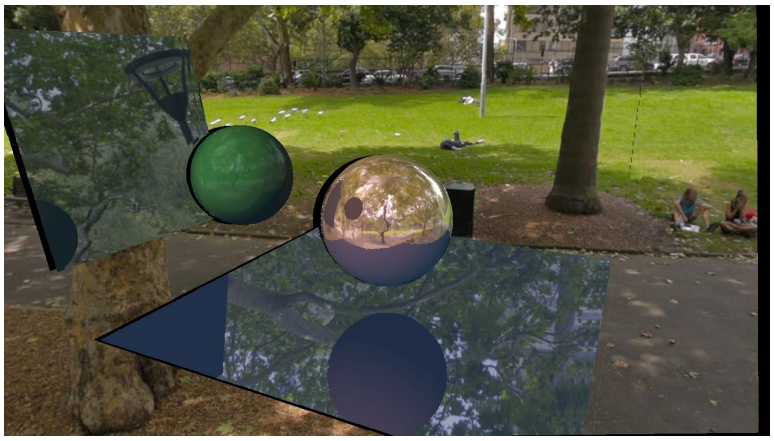


Abbildung 25: Ergebnis des Blendings

Sobald keine Bewegung in der Szene stattfindet, also beide Blickwinkel identisch sind, werden auf den reflektierenden Flächen nicht alle Pixel beschrieben (Abb. 26). Des Weiteren ist eine Unruhe bei Bewegung auf der Kugel zu erkennen. Auf einzelnen Bildern ist diese in Form von Rauschen zu sehen (Abb. 25).

Dass die Anzahl der gewählten, möglichen Schritte entscheidend ist, die in der gradientDescent-Methode genutzt werden, ist an den Abbildungen 27(a), 27(b) und 27(c) erkennbar. Der initiale Wert liegt bei 20 Schritten (Abb. 27(a)), dabei kann ab einem bestimmten Winkel nicht mehr die richtige Reflexion von der gradientDescent-Methode gefunden werden. Bei 40 Schritten hingegen (Abb. 27(b)), wird deutlich, dass mehr Reflexionen rekonstruiert werden können. Es wird deutlich, dass bei wachsender Anzahl der möglichen Schritte die Qualität steigt (Abb. 27(c)). Ab einer gewissen Größe, steigt die optische Qualität jedoch nicht weiter. Dann wird in der gradienDescent-Methode oft pro Schritt das benachbarte Pixel gewählt, wodurch weiterhin Rauschen auf der Kugel entsteht (Abb. 27(d) und 27(e)).

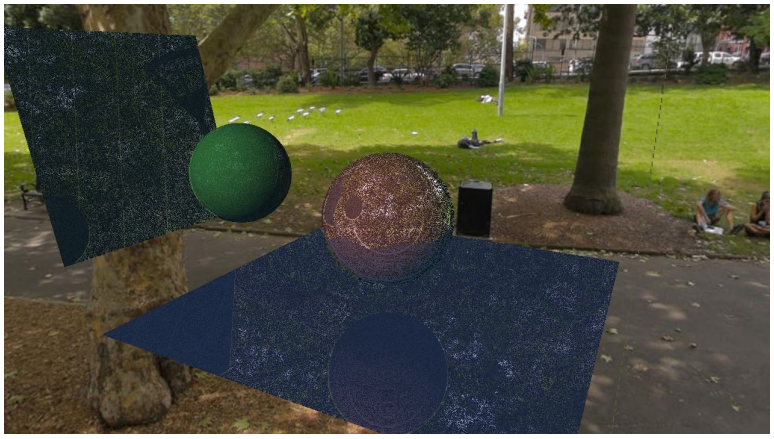
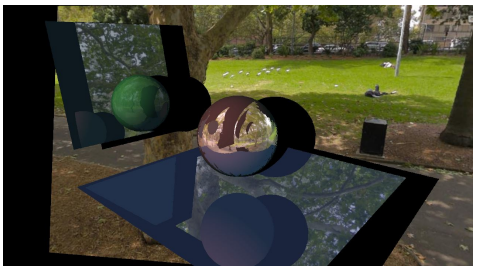
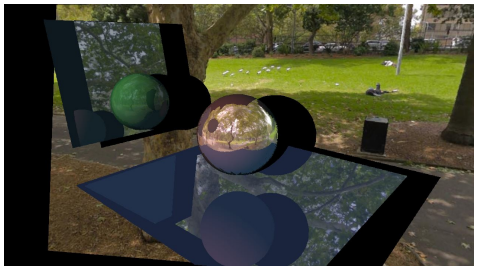


Abbildung 26: Bei identischen Blickwinkeln entstehen Löcher

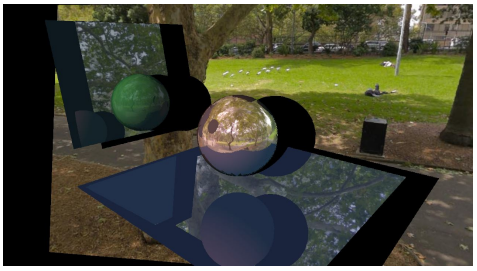
Mit steigender Schrittanzahl steigt auch die Rechenlast und die benötigten Hardwareressourcen. Pro Schritt finden drei Texturelookups statt. Jedoch ist bei hoher Schrittanzahl die Qualität des Ergebnisbild höher.



(a) GradientDescent-Methode mit 20 möglichen Schritten



(b) GradientDescent-Methode mit 40 möglichen Schritten



(c) GradientDescent-Methode mit 100 möglichen Schritten

(d) GradientDescent-Methode mit 280 möglichen Schritten

(e) GradientDescent-Methode mit 281 möglichen Schritten

Abbildung 27: Ergebnisse der gradientDescent-Methode mit unterschiedlicher Schrittanzahl

Ein weiterer negativer Aspekt der beim Blending deutlich wird, ist die Bestimmung der Reflexion auf unebenen Flächen. Sobald benachbarte Normalen zu große Unterschiede aufweisen, findet die gradientDescent-Methode kein Ergebnis und lässt das Ausgabepixel unbeschrieben. Zu beobachten ist dieses Verhalten an Abbildung 28. Auf dem Kopf des Stanfordbunnys ist statt der Reflexion an einigen Stellen nur die diffuse Farbe abgebildet. Das Modell musste zu Gunsten der Performanz eingeschränkt werden.

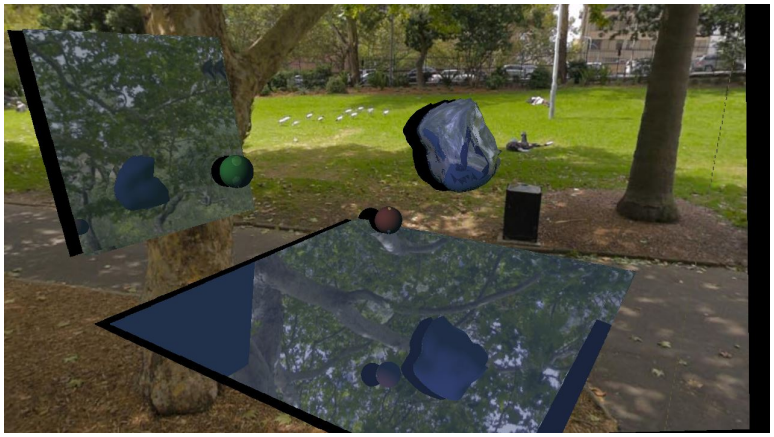


Abbildung 28: Problematik bei überkreuzten Normalen

3.2.3 Optimierung des Blendings

Die undefinierten Punkte, die bei Stillstand der Kamera auftreten, können mit einer Abfrage verhindert werden. Sobald die Ergebnisse der Qualitätsfunktion bei beiden möglichen Reflexionspositionen identisch sind, wird ein Ergebnis auf 1, das andere auf 0 gesetzt. Dadurch findet keine Interpolation statt und die richtige reflektierte Farbe liegt für jedes Pixel vor.
Durch diese Abfrage, die sich in der For-Schleife der gradientDescent-Methode befindet werden mehrere Verbesserungen erreicht:

```
1 if(bestQuality <= ensuredQualityThreshold) {
2 return outputCoord;}
```

Mit Hilfe der Abfrage wird die Suche nach einem besseren Ergebnis beendet, sobald die gewählte Qualität erreicht wurde (ensuredQualityThreshold). Dadurch wird die Möglichkeit auf ein noch exakteres Ergebnis nicht genutzt. Die Idee, diese Möglichkeit zu erhalten, indem der Schwellwert (Threshold) reduziert wird, liefert jedoch keine erkennbaren Verbesserungen in Hinblick auf die optische Qualität. Dieser vermeintliche Nachteil wurde damit entkräftet. Die Vorteile fallen dem Benutzer hingegen auf. Da die Funktion bei Finden eines Ergebnisses mit bestimmter Qualität abbricht, entsteht erkennbar weniger Rauschen bei Bewegung der Kamera. Das Hin- und Herspringen

zwischen benachbarten Pixeln wird damit umgangen. Des Weiteren wird der Algorithmus performanter. Vor der Abfrage wird die mögliche Anzahl von Schritten definitiv genutzt, auch wenn ein Ergebnis bereits gefunden wurde. Durch die Verbesserung wird die Suche „vorzeitig“ beendet. Somit entfallen pro nicht genutzter Iteration drei Texturelookups.
Ein Optimierungsansatz um das Problem bei unebenen Flächen mit unterschiedlichen Normalen zu lösen, ist das Anpassen der Normalen vor dem Warpingalgorithmus. Dazu wird ein zweiter Raytracepass genutzt. Im ersten Raytracepass werden nur die Normalen berechnet und als Textureingabe an den zweiten Raytracepass übergeben. Da die Normalen nun bereits vorliegen und nicht im eigentlichen Raytracer berechnet werden müssen, können die Normalen mit Berücksichtigung der Nachbarschaft bearbeitet werden[6]. In einer zusätzlichen Methode wird geprüft, ob sich benachbarte Normalen so unterscheiden, dass die gradientDescent-Methode keine Ergebnisse liefert. Sobald das der Fall ist, wird die aktuelle Normale geglättet. Damit soll erreicht werden, dass die Positionen der Reflexionen nicht so weit auseinander liegen. In der gradientDescent-Methode ist es dann wahrscheinlicher, dass ein Ergebnis gefunden werden kann. Die Glättung der Normalen muss vor der Entstehung der reflektierten Strahlen im Raytracepass passieren, damit von vornherein ein alternatives Ergebnis beim Raytracing entsteht. Die Glättung danach hätte zur Folge, dass das reproduzierte Ergebnis keine homogenen Reflexionen liefert.

4 Ergebnisse

4.1 Raytracing, Warping und Blending

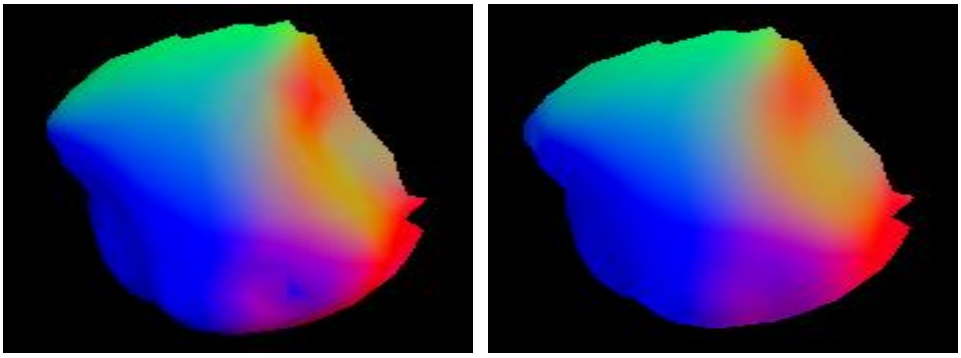
Die Parallelisierung des Raytracingalgorithmus, die durch die Grafikkarte ermöglicht wird, macht den Algorithmus echtzeitfähig. Jedoch muss eine Szene vorliegen, die nicht viel Geometrie aufweist. Der Testfall des Stanfordbunnykopfes ist nicht mehr in Echtzeit ausführbar. Des Weiteren ist für eine hohe Bildfrequenz das simple Beleuchtungsverfahren und das Fehlen von mehreren Primärstrahlen pro Pixel verantwortlich. Die vom Raytracer erzeugten Texturen sind für das Warpingverfahren geeignet. Sowohl in Hinblick auf die optische Qualität, als auch auf die mathematische Korrektheit können diese Texturen verwendet werden.
Besteht die Forderung auf Echtzeit und eine komplexe Szene mit mehr als 100 Dreiecken, müssen die Berechnungen auf ein rechenstarkes Remote-Rendering System übertragen werden.
Bei dem Warping von diffusen und reflektierenden Oberflächen gilt solch eine Einschränkung nicht. Beide Berechnungen passieren in Echtzeit. Durch die

[6] Code im Anhang

Aufteilung in Layer ist die korrekte Berechnung beider Fälle möglich. Dabei ist das Verfahren in jetziger implementierter Form solide. Bei der getesteten Szene kommen keine Probleme auf.
Herausforderungen entstehen hingegen beim Blending. Für Geometrie mit ebenen bis leicht gekrümmten Oberflächen liefert der Blendingalgorithmus überzeugende Ergebnisse. Bei angepasster Schrittanzahl für die gradientDescent-Methode werden auch für deutlichere Kamerarotationen zufriedenstellende Ergebnisse gefunden. Bei sehr unebenen Flächen weisen die Normalen deutliche Unterschiede zu ihren Nachbarn auf. In diesem Fall ist die optische Qualität des Ergebnisses nicht ausreichend. Viele Pixel bleiben unbeschrieben, weil die geforderte Qualität nicht gewährleistet wird.
Zusammenfassend gilt, dass für simple Szenen die Algorithmen solide Ergebnisse liefern, mit denen weiter gearbeitet werden kann.

4.2 Optimierung

Das Ergebnis der Optimierung des Reflective-Warping Verfahrens muss differenziert werden. Die Vermeidung von nicht beschriebenen Pixeln bei ruhender Kamera erzielt ein zufriedenstellendes Ergebnis. Ebenso kann die Rauschunterdrückung als erfolgreich eingestuft werden. Die optische Qualität wird beibehalten, während das Rauschen bei Kamerabewegungen reduziert wird. Außerdem werden Texturelookups vermieden, wodurch der Algorithmus performanter wird.
Die Glättung der Normalen ist ein Verfahren mit Potenzial. Es wird eine Steigerung der optischen Qualität verzeichnet, dabei treten jedoch an vorher unproblematischen Stellen im Bild Artefakte auf. Des Weiteren ist die Beobachtung aufgetreten, dass die Glättung eine bestimmte Größe der Geometrie erfordert. An den Abbildungen 29(a) und 29(b) wird deutlich, dass die Glättung der Normalen funktioniert, Übergänge werden weicher.



(a) Normalen vor der Glättung (b) Normalen nach der Glättung

Abbildung 29: Glättung der Normalen

Durch diese Glättung wird das Ergebnis des Blendings verbessert. Dabei

wurde eine Schrittanzahl von 100 in der gradientDescent-Methode gewählt. (Abb. 30(a) und 30(b)).

(a) Ergebnis vor der Glättung (b) Ergebnis nach der Glättung

Abbildung 30: Resultat der Glättung

Es wird jedoch auch deutlich, dass nicht alle Pixel beschrieben werden können und außerdem neue Artefakte entstehen. Das liegt an dem Glättungsverfahren. Für die Unterscheidung der einzelnen Objekte ist eine komplexere Lösung nötig, damit nicht zwischen Normalen interpoliert wird, bei denen keine geometrische Beziehung vorhanden ist. Diese Problematik wird anhand Abbildung 31 deutlich. Dort ist zu sehen, dass der grüne Anteil der Ebene mit den Normalen der Kugel vermischt wird.

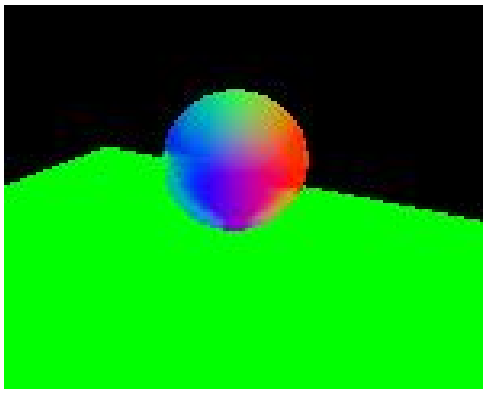


Abbildung 31: Problematik der objektübergreifenden Glättung

In der Glättungsmethode, die sich im Raytrcer befindet, wird über benachbarte Normalen iteriert. Abfragen, ob Nachbarnormalen dem Hintergrund angehören oder in einem zu großen Winkel zur betrachteten Normale stehen, entscheiden über die Gewichtung bei der Glättung. Eine weitere Abfrage pro Nachbarnormale soll bestimmen, ob die betrachtete und Nachbarnormale in die ähnliche Richtung zeigen. Ist dies nicht der Fall, wird das

Gewicht ebenfalls angepasst. Durch diese Abfragen sind deutliche Verbesserungen zu erkennen. Die Bereiche der Objekte, die zu verschiedene Normalen aufweisen, bleiben jedoch undefiniert. Deutlich sichtbar ist dies auf der kleinen Kugel, sowie am Rand des Kopfes vom Stanfordbunny in Abbildung 30(a).
Zusammenfassend wird festgehalten, dass die Glättung der Normalen im Raytracepass ein wirksamer Ansatz ist. Allerdings ist eine Weiterentwicklung notwendig, um auch komplexere Geometrien mit zufriedenstellendem Ergebnis bei dem Reflective-Warping Verfahren benutzen zu können. Außerdem zeigt die Glättung bei kleinen Objekten kaum Wirkung. Auch dafür ist eine gesonderte Behandlung nötig.

5 Fazit und Ausblick

Das Wachstum vom Einsatz von Remote-Rendering Systemen und Head-Mounted Displays erhöht die Nachfrage von Verbesserung dieser Systeme. Die dazukommende Möglichkeit auch reflektierende Oberflächen mit dem Warping zu nutzen, steigert die Einsatzmöglichkeit des Algorithmus. Durch Stabilisation der Bildfrequenz oder der Vermeidung der *Motionsickness* kann das Reflective-Warping zukünftig vermehrt erfolgreich eingesetzt werden.
Einschränkungen dieser Aussagen gelten jedoch auch. Für einen stabilen Algorithmus wird es nötig sein, das Verfahren zu ergänzen. Eine Möglichkeit kann die Kombination mit dem optischen Fluss ergeben. Dadurch kann es erreicht werden, die Objekte beim Glätten der Normalen zu unterscheiden. Davon kann die problematische Randbehandlung von Objekten profitieren. Generell wird eine komplexere Glättung der Normalen gefordert sein. Wie im Abschnitt der Ergebnisse deutlich wird, ist durch die Glättung eine Steigerung der optischen Qualität möglich. Damit die Qualität für jeden Fall erreicht wird, sind jedoch weitere Forschungen nötig.
Besonders bei Remote-Rendering Systemen oder bei zukünftig steigender Leistung der Hardware kann das Verfahren benutzt werden. Durch einen leistungsstarken Raytracer kann globale Beleuchtung ermöglicht werden. Die Limitierung von Dreiecken in der Szene kann dadurch ebenfalls reduziert werden.
Ansatzpunkte für Verbesserungen auf Seiten des Raytracers sind Beschleunigungsstrukturen oder ein Aufteilen der Szene, sodass pro Bild nur Teile der Szene neu gerendert werden. Außerdem kann das Verfahren um einen Layer erweitert werden. Gebrochenes Licht wird dann als hinzukommende Schicht behandelt und verringert die Distanz zur Realität.

Diese Arbeit hat folglich bestätigt, dass die Optimierung des Reflective-Warping Verfahrens realistisch umsetzbar ist. Das Glätten der Normalen ist ein erfolgversprechender Ansatz, der jedoch weiterhin Verbesserungen benötigt.

6 Anhang

```
1  vec4 adjustNormal() {
2
3  float stepHorizontal = 1.0 / resolution.x;
4  float stepVertical = 1.0 / resolution.y;
5  float sensitivity = 0.000001;
6
7          //original normal
8  vec4 norm = normalize(texture(normalTexture, (uv + 1) / 2));
9
10 float w1 = 1.0;
11 float w2 = 1.0;
12 float w3 = 1.0;
13 float w4 = 1.0;
14
15 vec4 nU = normalize(texture(normalTexture, (uv + 1) / 2 +
       vec2(0.0, stepVertical)));
16 vec4 nD = normalize(texture(normalTexture, (uv + 1) / 2 -
       vec2(0.0, stepVertical)));
17 vec4 nR = normalize(texture(normalTexture, (uv + 1) / 2 +
       vec2( stepHorizontal, 0.0)));
18 vec4 nL = normalize(texture(normalTexture, (uv + 1) / 2 -
       vec2(stepHorizontal, 0.0)));
19
20 vec4 savenU = nU;
21 vec4 savenD = nD;
22 vec4 savenR = nR;
23 vec4 savenL = nL;
24
25 if(length(norm-nU)<=0.0001 || length(norm-nD)<=0.0001 ||
       length(norm-nL)<=0.0001 || length(norm-nR)<=0.0001) {
26                 return vec4(norm);
27         }
28 else {
29   for(int i=1; i<=15; i++) {
30         nU = texture(normalTexture, (uv + 1) / 2 + vec2(0.0,
               i * stepVertical));
31         if(length(nU)<0.001) {
32                 w1 = 0.0;
33         }
34         if(length(savenU - nU) < sensitivity) {
35                 w1 = 0.0;
36         }
37         if(dot(norm, nU)<=0.1) {
38                 w1 = 0.25;
39         }
40
```

```
41          nD = texture(normalTexture, (uv + 1) / 2 - vec2(0.0,
                i * stepVertical));
42          if(length(nD)<0.001) {
43                  w2 = 0.0;
44          }
45          if(length(savenD - nD) < sensitivity) {
46                  w2 = 0.0;
47          }
48          if(dot(norm, nD)<=0.1) {
49                  w2 = 0.25;
50          }
51
52          nR = texture(normalTexture, (uv + 1) / 2 + vec2(i *
                stepHorizontal, 0.0));
53          if(length(nR)<0.001) {
54                  w3 = 0.0;
55          }
56          if(length(savenR - nR) < sensitivity) {
57                  w3 = 0.0;
58          }
59          if(dot(norm, nR)<=0.1) {
60                  w3 = 0.25;
61          }
62
63          nL = texture(normalTexture, (uv + 1) / 2 - vec2(i *
                stepHorizontal, 0.0));
64          if(length(nL)<0.001) {
65                  w4 = 0.0;
66          }
67          if(length(savenL - nL) < sensitivity) {
68                  w4 = 0.0;
69          }
70          if(dot(norm, nL)<=0.1) {
71                  w4 = 0.25;
72          }
73
74          norm = norm + nU*w1 + nD*w2 + nL*w3 + nR*w4;
75
76          savenU = nU;
77          savenD = nD;
78          savenR = nR;
79          savenL = nL;
80
81          w1 = 1.0;
82          w2 = 1.0;
83          w3 = 1.0;
84          w4 = 1.0;
85      }
86  }
```

```
87 return normalize(norm);
88 }
```

Literatur

[Abe09] Abert, Oliver: *Augenblick*. Eul Verlag, 2009

[App68] Appel, Arthur: Some techniques for shading machine renderings of solids. In: *Proceedings of the Spring Joint Computer Conference*, AFIPS, 1968. – Online erhältlich unter http://people.reed.edu/~jimfix/math385/lec01.1/Light/"-Appel.pdf Eingesehen am 29.3.2015

[BMS+12] Bowles, Huw ; Mitchell, Kenny ; Sumner, Robert W. ; Moore, Jeremy ; Gross, Markus: Iterative Image Warping. In: *Computer graphics forum* Bd. 31, EUROGRAPHICS, 2012, S. 237–246. – Online erhältlich unter http://www.disneyresearch.com/wp-content/uploads/Iterative-Image-Warping-Paper.pdf Eingesehen am: 02.5.2015

[DBB06] Dutré, Philip ; Bala, Kavita ; Bekaert, Philippe: *Advanced Global Illumination, 2. Edition*. A K Peters, 2006

[Hen11] Henrich, Niklas: *Verfahren zur globalen Beleuchtungsberechnung mit farbmetrisch korrekter Ausgabe*. Der andere Verlag, 2011

[Hon13] Honsdorf, Jonas: Path Tracing und dessen Optimierung durch Sampling, 2013. – Online erhältlich unter http://kola.opus.hbz-nrw.de/volltexte/2013/947/pdf/"-Ausarbeitung.pdf Eingesehen am: 8.4.2015

[Kaj86] Kajiya, James T.: The rendering equation. In: *Computer Graphics*, ACM SIGGRAPH, 1986. – Online erhältlich unter http://x86.cs.duke.edu/courses/cps124/compsci344/cps124/"-fall09/notes/16_rendering/p143-kajiya.pdf Eingesehen am: 2.4.2015

[LRR+14] Lochmann, Gerrit ; Reinert, Bernhard ; Ritschel, Tobias ; Müller, Stefan ; Seidel, Hans-Peter: Real-time Reflective and Refractive Novel-view Synthesis. In: *VMV 2014: Vision, Modeling & Visualization*, Eurographics Association, 2014, S. 9–16. – Online erhältlich unter http://resources.mpi-inf.mpg.de/SpecularWarping/"-SpecularWarping.pdf Eingesehen am: 10.4.2015

[MJ97] Mcmillan Jr., Leonard: An Image-Based Approach To Three-Dimensional Computer Graphics, 1997. – Online erhältlich unter http://citeseerx.ist.psu.edu/viewdoc/download?doi="-10.1.1.34.2524\&rep=rep1\&type=pdf Eingesehen am: 12.4.2015

[MJMB97] McMillan Jr., Leonard ; Mark, William R. ; Bishop, Gary: Post-Rendering 3D Warping. In: *In Proceedings of 1997 Symposium on Interactive 3D Graphics*, Department of Computer Science University of North Carolina at Chapel Hill, 1997, S. 7–16. – Online erhältlich unter http://www.csbio.unc.edu/mcmillan/pubs/I3D97_Mark.pdf Eingesehen am: 10.4.2015

[NRH+77] Nicodemus, F.E. ; Richmond, J.C. ; Hsia, J.J. ; Ginsberg, I.W. ; Limperis, T.: Geometrical Considerations and Nomenclature for Reflectance, U.S Department of Commerce, 1977. – Online erhältlich unter http://graphics.stanford.edu/courses/cs448-05-winter/"-papers/nicodemus-brdf-nist.pdf Eingesehen am: 8.4.2015

[PH10] Pharr, Matt ; Humphreys, Greg: *Physically Based Rendering From Theory to Implementation, Second Edition.* Morgan Kaufmann, 2010

[WV12] Walia, Ekta ; Verma, Vishal: A Computationally Efficient Framework for 3D Warping Technique, International Journal of Computer Graphics, 2012. – Online erhältlich unter http://www.sersc.org/journals/IJCG/vol3_no1/1.pdf Eingesehen am: 10.4.2015