

Spiegelblöcke in Minecraft

Simon Köhler, Johannes Wambach



Figure 1: Szene aus Minecraft mit mehreren Spiegeln

Abstract

In vielen existierenden Shadern für Minecraft existieren bereits Ansätze zu Spiegeleffekten. Diese beschränken sich jedoch auf Wasserreflexionen. Ein Shader für Spiegeleffekte von regulären Blöcken liegt noch nicht vor.

In dieser Arbeit wurde mit Minecraft als Shader-Engine das Verfahren Screen-Based-Raytracing für die Darstellung spiegelnder Blöcke umgesetzt. Für verschiedene weitere Effekte wurden lokale Verfahren, wie Kantendetektoren, Farbfilter, Weichzeichner eingesetzt. Diese Effekte wurden auf verschiedene Blocktypen projiziert, um diese in der Minecraft-Welt gegenüberzustellen.

Der entstandene Shader hat Umfang, Qualität und Robustheit, die genügen diesen in der Minecraft-Community zu veröffentlichen. Die Einschränkung der Spiegelbilder auf die Abbildungen des Bildschirms jedoch können unter Umständen bei Nutzern zu eingeschränkter Zustimmung führen.

Keywords: minecraft, raytracing, mirror

Concepts: •Computer Graphics → Shader-Entwicklung; Lokale Operatoren;

1 Einleitung

Im Rahmen des Praktikums "Shader Konzepte für Spieleentwicklung" wurde unter Verwendung des Computerspiels Minecraft ein Shader entwickelt, der es ermöglicht spiegelnde Objekte in der Welt zu platzieren. In dieser Arbeit werden die Grundlagen, Realisierungsschritte und Ergebnisse in folgenden Kapiteln erläutert:

- I. Einleitung
- II. Grundlagen - beleuchtet die Verfahren Ray-Tracing, Screen-space-Ray-Tracing, Kantendetektoren und weitere lokale Verfahren.
- III. Konzept - beschreibt den vollständigen Rendering-Prozess
- IV. Realisierung - erläutert die verwendeten Technologien und Realisierungsschritte
- V. Ergebnisse - beschreibt die Resultate des Projekts mit Bildern

VI. Zusammenfassung und Ausblick - skizziert mögliche weitere Entwicklungsschritte und angrenzende Fachgebiete und Forschungsergebnisse

Zu Beginn des Projekts wurde geplant, welches Ziel zur Abgabe erreicht werden soll. Es soll dem Nutzer ermöglicht werden, beliebige Blöcke eines vordefinierten Typs in der Welt zu platzieren, die eine Spiegelung der Umgebung zeigen. Des weiteren sollen verschiedene Typen von Blöcken unterschiedliche Spiegeleigenschaften aufweisen, um weiterführende Rendering-Konzepte zu illustrieren. Die Berechnung von Spiegelungen ist eine der komplexesten Aufgaben in der Computer Graphik. Das meist genutzte Verfahren ist das Reflection Mapping, auch Environment Mapping oder Cube Mapping, bei dem eine Textur als Reflexion auf Objekte gemappt wird (vgl. [Blinn and Newell 1976]). Unter anderem gibt es auch die Möglichkeit, besonders bei Wasserspiegelungen, die komplette Szene zusätzlich auf den Kopf gestellt zu rendern um so die Spiegelung zu erhalten. Das einzige richtige Verfahren für Reflexionsberechnung ist das Ray-Tracing, allerdings ist es für Echtzeitanwendungen meist zu aufwendig. Wir wollten einen Algorithmus der nur mit den Informationen von einem deferred renderer (G-Buffer) Reflexionen in Echtzeit berechnen kann. Daher haben wir das Konzept des Screen-Space Ray-Tracing implementiert [SOUSA and SCHULZ 2011][McGuire and Mara 2014].

2 Grundlagen

2.1 Kantendetektoren

In vielen graphischen Anwendungsgebieten werden Algorithmen benötigt, die Kanten und Ecken von Objekten auf einem zweidimensionalen Bild erkennen können. Nicht ohne Grund gehört Kantendetektion zu den wichtigsten menschlichen Sehfähigkeiten (s. [van Vliet et al. 1989], S. 169). Sie hilft Objekte optisch und logisch von ihrer Umgebung zu trennen.

Kantendetektoren sind lokale Verfahren, die anhand von diskreten Informationen eines Punktes und seiner Nachbarn operieren. Sehr einfach gefasst könnten von einem Punkt die Farben der 4- oder 8-Nachbarschaft subtrahiert werden. Der absolute Wert der Differenz stellt damit den summierten Unterschied von P zu seinen Nachbarn dar. (vgl. [van Vliet et al. 1989], S. 170). Die Nachbarschaften werden in der Regel mit Matrizen dargestellt. Matrizen für 4-Nachbarschaft (M_4) und 8-Nachbarschaft (M_8) können beispielsweise wie folgt gewählt werden:

$$M_4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, M_8 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Die Verwendung dieser Matrizen wird auch als Laplace-Filter bezeichnet. Sind die behandelten Werte W auch in einer 3×3 -Matrix dargestellt, kann das Resultat R für den zentralen Punkt wie folgt - am Beispiel mit einer 4-Nachbarschaft - errechnet werden:¹

$$W = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{pmatrix}$$

$$M = W \times M_4 = \begin{pmatrix} 0 & w_2 & 0 \\ w_4 & -4w_5 & w_6 \\ 0 & w_8 & 0 \end{pmatrix}, R = \sum_{n=1}^9 M_n$$

In der Regel werden Kanten detektoren für direkte visuelle Akzentuierungen auf den lokalen Grauwerten berechnet. Darüber hinaus können auch andere lokale Informationen, wie beispielsweise die Entfernung zum Betrachter verwendet werden. Auf diesem Weg, kann die Differenz der Entfernung zwischen betrachtetem Punkt und Nachbarschaft errechnet werden.

2.2 Gaußscher Blur

Unterstützend für mehrere umgesetzte Verfahren wurde zudem der Gaußsche Weichzeichner (engl. Gaussian Blur) verwendet. Dieses lokale Verfahren mischt die Farben in einer Nachbarschaft liegender Punkte beispielsweise mit einer 5×5 -Matrix A_G . Für einen Standardweichzeichner gilt

$$A_G = \frac{1}{273} \times \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

Das verwendete Bild wirkt nach der Anwendung des gaußschen Weichzeichner verwaschen und ist unscharf. Der Weichzeichner kann jedoch in Bildern mit scharfen Kanten, die durch lineare Algorithmen entstanden sind, verwendet werden, um ein realistischeren Eindruck zu erzeugen.

2.3 Weitere lokale Verfahren

Um Effekte ähnlich zu alten Fotografien zu erreichen, können lokale Verfahren ohne Einbeziehung der Nachbarschaft verwendet werden. Diese Verfahren berechnen Farbwerte auf Basis der drei Farbkanäle Rot, Grün und Blau. Eine Farbe F mit $F = (R, G, B)$ wird mit einer Matrix O vektorweise multipliziert. Zwei Matrixoperatoren sind Graustufe G und Sepia S mit:

$$G = \frac{1}{3} \times \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, S = \begin{pmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{pmatrix}$$

¹Der Operator \times beschreibt hier die einfache komponentenweise Matrix-Matrix-Multiplikation

Die Farbe des resultierenden Pixel F_G und F_S ergibt sich wie folgt:²

$$F_G = G \times F$$

$$F_S = S \times F$$

Die Matrixoperatoren werden bei der Anwendung pixelweise mit der Farbe multipliziert und so modifiziert. Ein Graustufen-Bild enthält nach Anwendung von G nur noch Grauwert-Farben und ist somit schwarz-weiß. Nach Anwendung des Sepia-Operators bekommt das Bild einen Rot-Gelb-Braun-Stich und sieht nach einem altem verblichenen Foto aus.

2.4 Ray-Tracing

Ray-Tracing ist grundsätzlich ein Beleuchtungsverfahren für die Visualisierung von 3D-Welten in einem 2D-Bild. Die Grundlage für Ray-Tracing sind sogenannte Seh-Strahlen, welche die Blickrichtungen beziehungsweise das Sehfeld der Kamera simulieren sollen. Beim sogenannten Backward-Ray-Tracing³ werden die Sehstrahlen beginnend an der Kamera in die Szene gerichtet und mithilfe der Objekte in der Welt umgeleitet (s. Figure 5).

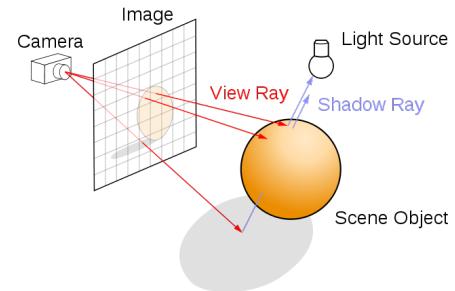


Figure 2: Ray-Tracing Algorithmus, By Henrik GFDL, <https://commons.wikimedia.org/w/index.php?curid=3869326>

Die Reflektionen oder Spiegelungen an den Objekten folgen verschiedenen Heuristiken, die abhängig von der Oberfläche und Struktur der Objekte sind. Glatte Objekte spiegeln dann beispielsweise einfach den Sehstrahl nach dem Prinzip *Einfallswinkel = Ausfallwinkel*. Bei matten, halbspiegelnden oder semi-durchlässigen Objekten bestehen zusätzlich weitere Abwandlungen bei der Spiegelung. So können bei einer matten Oberfläche aus einem Sehstrahl mehrere Sehstrahlen entstehen, die in verschiedene Richtungen gestreut werden. Bei halbspiegelnden Objekten würde der gespiegelte Sehstrahl mit geringerer Intensität gewichtet. Für die Berechnung der Lichtstärke und Farbe eines Pixels werden ein oder mehr Sehstrahlen in die Welt gesendet die nach möglichen Reflexionen entweder eine Lichtquelle treffen oder durch eine Fläche absorbiert werden. Die Summe der Leuchtkräfte der getroffenen Lichtquellen beschreibt somit die Helligkeit eines Pixels (vgl. [Glassner 1989] S. 8-9).

Die Farbe des Pixels wird durch die Farbe der Lichtquelle in Verbindung mit den Farben der reflektierenden Oberflächen ermittelt. Die dabei beteiligten Farben werden subtraktiv zusammengeführt. Ein Seh-Strahl der an einer roten Oberfläche reflektiert

²Der Operator \times beschreibt hier die einfache komponentenweise Matrix-Vektor-Multiplikation

³Das sogenannte Forward-Ray-Tracing ist ein weiteres Ray-Tracing-Verfahren. Es bildet die physikalische Realität bezüglich beweglicher Photonen besser ab, da hier nicht Seh-Strahlen in die Welt geschickt werden, sondern Lichtstrahlen von allen bestehenden Lichtquellen in die Welt gesendet werden. (vgl. [Glassner 1989], S. 7)

und eine weiße Lichtquelle trifft wird als rot wahrgenommen, da die Farbspektren für grün und blau von der roten Oberfläche absorbiert werden. Ein Seh-Strahl, der nach Reflexion an einem grünen Objekt eine rote Lichtquelle trifft wirkt schwarz, da das vom Licht ausgehende rote Licht vollständig von der grünen Oberfläche absorbiert wird.

2.5 Screen-Space Ray-Tracing

Ray-Tracing-Verfahren sind im Grunde bezüglich Sichtweiten und Objektabbildungen nicht auf die Größe des zu rendernden Bilds beschränkt. Für die Kollisionsberechnungen sind grundsätzlich jegliche Informationen über Lichtquellen, Objektgeometrien und Objekteigenschaften sowie Kameraeigenschaften notwendig. Liegen diese Informationen in Teilen nicht vor, kann der Algorithmus auch nur eingeschränkt umgesetzt werden. Bei fehlenden Lichtinformationen ist beispielsweise die Helligkeitsberechnung nicht möglich und es muss von einer gleichmäßig beleuchteten Szene ausgegangen werden. Ein viel schwerwiegenderes Problem stellt das Fehlen von Objektgeometrien dar. Alle Strahlen, die den relativen Sichtbereich verlassen sind nicht weiter zu verfolgen und können nicht berechnet werden (s. Figure 3).



Figure 3: Einschränkungen des Screen-Space-Ray-Tracings. Ausschnitt aus Minecraft mit einem Spiegel auf der linken Seite. Der abgedunkelte Bereich ist das Spiegelbild. Der weiße Rand konnte nicht gespiegelt werden. Alle Informationen, die gespiegelt werden müssten, jedoch außerhalb des Sichtfelds liegen, können nicht gespiegelt werden.

Das Berechnen von Bildinformationen auf Basis des Bildes (Screen) selbst nennt sich Screen-Space Ray-Tracing. Wie oben beschrieben können Strahlen, die den Bildbereich verlassen, nicht weitergerechnet werden. Somit können beispielsweise Spiegelbilder nur eingeschränkt errechnet werden, da Bildinformationen nur für einen kleinen Teil der Welt vorliegen.

3 Konzept

Da in Minecraft eine Vielzahl von Blöcken existiert, besteht die Möglichkeit, für verschiedene Blocktypen unterschiedliche Spiegelungseffekte umzusetzen. Das im folgenden beschriebene Rendering-System unterstützt sowohl realistische Spiegelungseffekte als auch nicht-fotorealistisches Rendering. Cel-Shading ist ein berühmtes Verfahren, um ein Bild bewusst nicht-fotorealistisch sondern kunstvoll wirken zu lassen. Dagegen wird beim Physically based Rendering versucht Spiegelungen so an die Materialbeschaffenheiten anzupassen, um jegliche Art von Spiegelung physikalisch korrekt darzustellen.

Die Berechnungsschritte unseres Rendering-Systems beinhalten:

1. Berechnung der Reflektion mit Screen-Space Ray-Tracing
2. Anwendung eines Gaußschen Blur auf die Reflektionen
3. Vermischung von Textur und Reflektion
4. Berechnung der künstlichen Doppelreflektion
5. Berechnung und Anwendung weiterer Effekte

3.1 Screen-Space Ray-Tracing

Die Idee beim Screen-Space Ray-Tracing ist es mit Hilfe einer depth map die fehlenden Informationen zu rekonstruieren. Eine depth map ist eine Textur, die die Tiefeninformation eines Bildpixels speichert. Die Tiefe beschreibt hierbei die Entfernung von Objektoberflächen aus der Sicht der Kamera. Es wird nur die Distanz zwischen der Kamera und dem nahegelegendsten Objekt gespeichert. Eine depth map ermöglicht es von der Kamera aus Strahlen in die Szene zu schicken und diese auf Kollisionen mit Objekten zu testen.

Wir benutzen Ray-Tracing ausschließlich um Reflexionen zu berechnen. Daher führen wir nur einen Ray-Cast durch, wenn das entsprechende Fragment zu einem spiegelnden Block gehört. Diese Information speichern wir in einer entity map, die die Beziehung des Blocktyps zu den einzelnen Bildpixelen enthält. Um die Reflexionsfarbe eines Pixels zu berechnen wird ein Strahl von der Kamera zum entsprechenden Fragment geschickt und mit dem Normalenvektor dieses Fragments der Reflexionsvektor berechnet. Der Punkt der ersten Kollision eines Objektes mit dem Reflexionsvektors bestimmt nun die Farbe des zu berechnenden Pixels.

Mit folgender Formel lässt sich aus 2D-Texturkoordinaten uv und der inversen der Projektionsmatrix M_P^{-1} eine 3D-Position p im Camera Space berechnen:

$$p = M_P^{-1} \times \begin{pmatrix} u * 2 - 1 \\ v * 2 - 1 \\ depth \\ 1 \end{pmatrix}$$

Dabei ist $depth$ die Tiefe des Fragments aus der depth map. Zusätzlich müssen die xyz-Koordinaten von p noch durch seine w-Koordinate geteilt werden. Der normalisierte Vektor von p entspricht nun der Kamerarichtung. Mit der Normalen aus der normal map und dieser Richtung kann nun der Reflexionsvektor berechnet werden.

Um die Kollision von Objekten mit dem Reflexionsvektor zu berechnen, verwenden wir das Verfahren des Ray-Marching. Dabei wird ein Strahl im Camera Space von p mit Richtung des Reflexionsvektors in die Szene geschickt und in bestimmten Abständen auf Kollision geprüft. Beim Kollisionstest werden zwei Tiefenwerte verglichen. Der erste Wert entspricht der z-Koordinate des Strahls. Der Sample-Punkt auf dem Strahl wird vom Camera Space zum Screen Space transformiert, um mit den erhaltenen Texturkoordinaten aus der depth map den zweiten Tiefenwert auszulesen. Wenn eine Kollision ermittelt wurde, kann der Algorithmus und das Ray-Marching abgebrochen werden.

Das Ergebnis des Screen-Space Ray-Tracing Algorithmus wird in einer Textur gespeichert, die für jeden Pixel angibt ob eine Reflexion berechnet werden konnte. Falls das Ray-Tracing erfolgreich war, werden noch die Texturkoordinaten des Ergebnispixels gespeichert. Somit wird es für spätere Effekte möglich für jeden spiegelnden Bildpixel die entsprechenden Reflexionspunkte auszulesen.

3.2 Glossy Reflections

Ein Effekt bei Spiegelungen sind unscharfe oder verschwommene Reflexionen. Beim klassischen Ray-Tracing wird hierzu nicht nur ein Reflexionsvektor in die Szene geschickt sondern mehrere. Die endgültige Farbe wird durch das Mitteln und gegebenenfalls Gewichten der Vektoren erhalten. Die Vektoren werden dabei

kegelförmig um den Reflexionsvektor erzeugt. Der Kegel gibt die maximale Abweichung von der idealen Reflexion und somit die Unschärfe der Spiegelung an.

Da wir beim Screen-Space Ray-Tracing auf die Bildinformationen beschränkt sind, verwenden wir anstatt Ray-Traced Blur einen Screen-Space Blur. Es wird also das Ergebnis des Ray-Tracing benutzt, um mit einem Gaußschen Blur die Reflexionen zu verschmieren. Es ist zu beachten, dass der Blur nicht auf die vom Ray-Tracing berechneten Spiegelungen angewendet werden kann. Der Fehler entsteht am Rand der Spiegelungen, da hier die Informationen für einen korrekten Blur nicht existieren. Daher wird für jeden spiegelnden Bildpixel die entsprechenden Reflexions-Texturkoordinaten ausgelesen und erst auf diese der Gaußsche Blur angewendet.

Eine Performance-Optimierung ohne Qualitätsverlust ist den einfachen Gaußschen Blur zu separieren in einen horizontalen und vertikalen Filter. Dies ist möglich da jede Faltungsmatrix des Gaußschen Blurs als dyadisches Produkt von zwei Vektoren ausgedrückt werden kann. Ein Produkt zwischen einem Spaltenvektor und einem Zeilenvektor wird auch äußeres oder dyadisches Produkt genannt. Die beiden Vektoren haben bei einem Gaußschen Filter die gleichen Werte.

$$A_G = \frac{1}{273} \times \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}, u = v = \begin{pmatrix} 0.061 \\ 0.242 \\ 0.383 \\ 0.242 \\ 0.061 \end{pmatrix}$$

Die Matrix A_G eines Gaußschen Blur kann auch durch das dyadische Produkt der Vektoren u und v berechnet werden [Filip 2014]:

$$A_G = v \otimes u$$

Dadurch lässt sich die Komplexität von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n)$ für jeden Pixel reduzieren, wobei n die Größe des Filterkernels ist.

Allerdings ist zu beachten, dass der vertikale Blur Filter nur auf die mit dem horizontalen Filter bearbeitete Reflexion des spiegelnden Blocks angewendet werden kann. Dies bedeutet, dass wir am Rand der Reflexion einen Fehler beim vertikalen Blur berechnen, da hier die Textur des Blockes oder die nähere Umgebung fälschlicherweise mit ein berechnet werden. Dieser Fehler ist allerdings vernachlässigbar. Der Gaußsche Weichzeichner wird bei den Reflexionen besonders bei schwachen Spiegelungen angewendet, um eine rauere Materialoberfläche zu berücksichtigen.

3.3 Physically Based Rendering

Ein Verfahren das besonders bei heutigen Spielen immer mehr zum Einsatz kommt, ist das Physically based Rendering. Dabei handelt es sich nicht um einen wohl definiertes Algorithmus, sondern kann eher als eine Reihe von Anforderungen an den Renderer gesehen werden [Russel J]. Ein physically based Shader sollte also bestimmte physikalische Konzepte und Gesetze berücksichtigen und zumindest annähernd umsetzen.

Ein solches Konzept ist Energieerhaltung, das aussagt, dass die Lichtmenge, die eine Oberfläche verlässt, nicht größer sein darf als die Lichtmenge, die auf die Oberfläche aufgetroffen ist. Des Weiteren gilt, dass Reflexion und Diffusion sich gegenseitig ausschließen. Dies hat zur Folge, dass eine perfekt spiegelnde Oberfläche keine Streuung des Lichtes verursachen kann und umgekehrt. Daraus lässt sich der Parameter Reflectivity ableiten, der das Reflexionsvermögen eines Objektes angibt:

In unserem Rendering-System kann die Reflectivity pro Blocktyp gesetzt werden. Unter Berücksichtigung der Energieerhaltung und der Reflectivity R wird die Farbe der Textur C_t mit der Farbe der



Figure 4: Reflectivity, http://www.marmoset.co/wp-content/uploads/pbr_theory_conservation.png

Reflexion C_r zu der endgültigen Farbe C_f vermischt:

$$C_f = C_t * (1 - R) + C_r * R$$

Ein weiterer Parameter ist Glossiness, auch Roughness oder Bluriness, der angibt wie rau die Oberfläche des Objektes ist. Je rauer die Oberfläche desto verschwommener ist die Spiegelung. Dabei spielt die Energieerhaltung wieder eine Rolle, weil bei rauen Oberflächen die Reflexion weiter gestreut wird und trüber ist, jedoch bei glatten Materialien die Spiegelung konzentrierter und heller ist. Diesen Effekt setzen wir mit dem Gaußschen Blur um. Der entsprechende Parameter für Glossiness heißt bei uns Blur Strength:

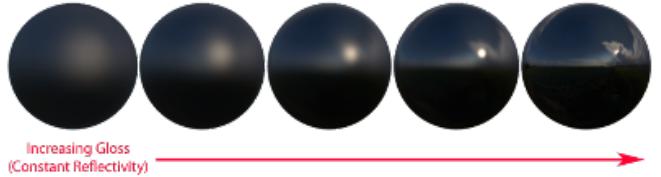


Figure 5: Blur Strength/Glossiness, http://www.marmoset.co/wp-content/uploads/pbr_theory_micro_cons.png

Ein letzter Effekt von Physically based Rendering, den wir umgesetzt haben ist der Fresnel-Effekt [Hoffman 2013]. Eine viel genutzte Annäherung an die Berechnung des Fresnel Terms ist Schlick's Approximation ([Schlick 1994], S. 7):

$$F_\lambda(u) = f_\lambda + (1 - f_\lambda)(1 - u)^5$$

Die Variable u ist der Winkel zwischen dem einfallenden Licht und der Normale der Oberfläche. Wegen $Einfallswinkel = Ausfallswinkel$ gilt bei uns also:

$$u = \cos\theta = (N \cdot V)$$

Wobei V die Kamerarichtung ist und N die Oberflächennormale ist. Es gilt $F_\lambda(0) = 1$ und $F_\lambda(1) = f_\lambda$. Somit gibt f_λ einen Minimalwert und der Term $(1 - f_\lambda)$ einen Maximalwert für den Fresnel Effekt an. Der Fresnel Effekt sorgt dafür, dass bei bestimmten Winkeln Reflexionen hinzugefügt werden. Dieser Effekt ist am schwächsten wenn der Betrachter frontal auf ein Objekt sieht und am stärksten wenn der Betrachter seitlich auf eine Oberfläche sieht. In unserem Rendering-System benutzen wir außerdem den Parameter Fresnel Power, der den Übergang zwischen Minimal- und Maximalwert des Fresnel Terms angibt. Bei steigender Fresnel Power P_f müssen Oberflächen zunehmend von der Seite betrachtet werden, um den Fresnel Effekt zu betrachten:

$$F_\lambda(u) = f_\lambda + (1 - f_\lambda)(1 - (N \cdot V))^{P_f}$$

Allerdings benutzen wir den Fresnel Effekt, um nicht korrekte Reflexionen auszublenden. Unser Fresnel Effekt sorgt dafür, dass

wenn der Fresnel Effekt am schwächsten ist Spiegelungen abnehmen und wenn Fresnel am stärksten ist verändern wir die Spiegelungen kaum. Dazu multiplizieren wir den Fresnel Term mit dem Reflectivity Parameter. Die annähernd korrekte Art Fresnel in unser System hinzuzufügen, wäre den Fresnel Term mit dem Reflectivity Parameter zu vermischen.

3.4 DoppelSpiegelung

Bei unserer Implementierung zum Screen-Space Ray-Tracing lassen wir einen Strahl nur einmal reflektieren. Somit ist die Berechnung von DoppelSpiegelungen bisher nicht berücksichtigt. Wir haben nun einen Trick angewandt, um dennoch auch diese Art der Reflexion zu unterstützen. Der Algorithmus wird auf das fertig gerenderte Bild angewandt. Es wird bei jedem Bildpixel überprüft, ob dieser zu einer Reflexion und der Reflexionspunkt selber auch zu einer Reflexion gehört. Wenn das der Fall ist, übernehmen wir die Farbe des Reflexionspunkts und vermischen diese nochmal mit der Textur. Im Prinzip wird also ein fertig berechneter Spiegelblock in den original zu berechnenden Spiegelblock kopiert (s. Figure 6). Dabei ist zu beachten, dass DoppelSpiegelungen den Blur Effekt ignorieren.



Figure 6: Links: Ohne DoppelSpiegelung, Rechts: Mit DoppelSpiegelung

3.5 Weitere Effekte

Ein interessanter Effekt bei Spiegelungen sind gebogene Spiegel. Eine Möglichkeit diesen Effekt zu realisieren ist es die Normale von einem Block zu modifizieren. Diese Transformation basiert auf der Position des Fragments auf dem Block, die auf einer Seite des Blocks sich von null bis eins in x- und y-Richtung erstrecken sollte. Die lokalen Texturkoordinaten, also die Texturkoordinaten der Vertizes, eines Blocks eignen sich hierfür nur durch Modifikation. Bei vielen Spielen, inklusive Minecraft, wird ein Texture-Atlas verwendet. Ein Bild, das viele kleinere Bilder enthält, welche jeweils eine eigene Textur darstellen, wird auch Texture-Atlas genannt. Es müssen also die Texturkoordinaten im fragment shader so angepasst werden, dass sich die Koordinaten von null bis eins in x- und y-Richtung erstrecken. Mit diesen Informationen kann nun die Normale entsprechend ihrer Position auf dem Block transformiert werden, um der Reflexion eine Beugung zu verleihen (s. Figure 7).



Figure 7: Spiegelung Links: nach außen gebogen, Rechts: nach innen gebogen

Der Celshading-Effekt wurde mithilfe von Kantendetektion auf der Tiefen-Map umgesetzt. Liegen neben dem betrachteten Punkt mehrere Punkte, die deutlich abgesetzte Tiefegrade besitzen, wird der Punkt dunkel bis schwarz eingefärbt. Für die Differenzbildung wurde folgende Matrix verwendet:

$$M = \begin{pmatrix} -1 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 2 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & 0 & -1 \end{pmatrix}$$

Für jeden spiegelnden Punkt wurde die Rohfarbe des Spiegelbilds mit einem Farboperator verrechnet. Dabei wird der Punkt nur schwarz eingefärbt, wenn die Differenz einen Schwellwert überschreitet.

Die Sepia- und Schwarz-Weiß-Effekte werden mit den im Abschnitt 2.2 beschrieben Operatoren umgesetzt. Dadurch erhalten die Spiegelbilder der jeweiligen Blöcke den Sepia- oder Schwarz-Weiß-Effekt.

4 Realisierung

Eine Problem beim Screen-Space Ray-Tracing sind die Ungenauigkeiten der depth map und der Kollisionserkennung. Eine Methode die Reflexionsqualität zu erhöhen ist das normal offset. Diese stammt ursprünglich vom Shadow Mapping Algorithmus und soll dort die sogenannte Shadow Acne, falsche Selbstschattierung, beheben [Holbert 2011]. Da Screen-Space Ray-Tracing sehr ähnlich zum Shadow Mapping funktioniert, funktioniert der normal offset auch hier. Dazu wird auf die Startposition des Reflexionsvektors p eine mit $Bias$ gewichtete Normale N addiert:

$$p = p + N \cdot Bias$$

Die Position wird also in Richtung der Normale der Oberfläche verschoben. Den Bias haben wir relativ groß mit 0.5 gewählt, so dass die Reflexion selber verschoben wird und somit nicht mehr korrekt ist (s. Figure 8). Allerdings hilft der normal offset eine robuste und nicht zerstückelte Reflexion zu erzeugen.



Figure 8: Oben: Ohne Normal Offset, Unten: Mit Normal Offset

Eine weitere Technik die Qualität und Performance der Spiegelungen beim Screen-Space Ray-Tracing zu erhöhen ist die Verfeinerung der Kollisionserkennung. Während eine Kollision

gesucht wird, wird, ähnlich dem Ray-Marching, die Szene vom Punkt auf dem Spiegel aus mit Richtung des Reflexionsvektor schrittweise abgetastet. Dabei nimmt die Schrittgröße zunehmend zu, um in relativ kurzer Zeit eine Kollision erkennen zu können. Wenn eine Kollision erkannt wird, wird der letzte Schritt zurückgegangen und die Schrittgröße wieder verkleinert. Diese Verfeinerung kann mehrmals durchgeführt werden, um eine möglichst genaue Kollisionserkennung zu erreichen.

Als letztes Verfahren, um besonders die Einschränkung der fehlenden Bildinformationen (s. Figure 3) zu kaschieren blenden wir die Reflexion in diesen Fällen zunehmend aus. Desto weiter sich ein Reflexionspunkt am Bildrand befindet, desto niedriger setzen wir den Reflectivity Parameter. So bekommen wir einen weichen Übergang zwischen Reflexion und Textur (s. Figure 9).



Figure 9: "Smooth Border" - Weicher Übergang (hier übertrieben)

Eine letzte Einschränkung des Screen-Space Ray-Tracing entsteht wenn Reflexionsstrahlen hinter Objekten verlaufen und somit eine Kollision verursachen. Diese Fälle kann unser Algorithmus erkennen. Das Problem hierbei ist allerdings, dass die Informationen für eine Spiegelung fehlen. Mit einem Spiegel ist es möglich um Ecken von Objekten zu sehen. Da der Betrachter ohne Spiegel diese Informationen nicht besitzt, sind sie in der depth- und color map auch nicht enthalten. Wir haben uns dazu entschieden in diesen Fällen keine falschen Reflexionen zu berechnen, sondern einfach die Textur anzuseigen. In Figure 10 ist dieses Problem sehr gut an den gelben Stellen in der Spiegelung zu sehen. Dieser Fehler lässt sich mit normal offset zumindest bei bestimmten Winkeln erheblich reduzieren.



Figure 10: Um die Ecke schauen verboten - normal offset = 0.2

5 Ergebnisse

Der entstandene Shader bietet mit verschiedenen Blöcken die Möglichkeit, verschiedene Grafikeffekte zu verdeutlichen. Die Resultate können auf den Abbildungen 11, 12, 13, 14, 15 nachvollzogen werden.

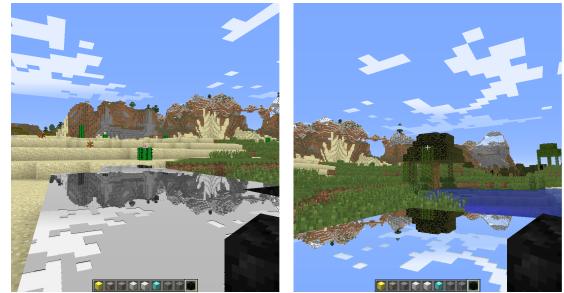


Figure 11: Schwarz-Weiß-Effekt (links) und klarer Spiegel (rechts)



Figure 12: Sepia-Effekt im Spiegel



Figure 13: Von links nach rechts zunehmender Blur



Figure 14: Von links nach rechts abnehmende Reflectivity



Figure 15: PBR in Aktion

6 Zusammenfassung

Innerhalb der vorgegebenen Bearbeitungszeit ist eine vollwertige Shader-Erweiterung für die Minecraft-Optifine-Engine entstanden, welcher das Potenzial besitzt in der Community veröffentlicht zu werden. Die spiegelnden Blöcke ermöglichen großartige Ausblicke und können für tolle minecraft-architektonische Projekte verwendet werden.

Weiteren Entwicklern, die eine Erweiterung der Mod programmieren möchten, steht es offen, beliebige Blöcke mit Spiegelungen zu versehen oder die bestehenden Effekte zu ändern, anzupassen oder auszutauschen. Darüber hinaus bestehen viele Möglichkeiten der Parametrisierung, die im Ausblick in Minecraft-Optionen transferiert werden können.⁴

Es gibt zahlreiche Möglichkeiten Screen-Space Ray-Tracing zu verbessern. Eine davon ist es das Ray-Marching auf den 2D-Raum anzupassen, sodass es kein Over- und kein Undersampling gibt [McGuire and Mara 2014]. Außerdem gibt es die Möglichkeit auf die Startpositionen der Strahlen einen jitter anzuwenden, um Artefakte zu kaschieren [Mara et al. 2014].

Acknowledgements

An die Modder von Optifine für die Bereitstellung einer OpenGL-GLSL-Shader-Engine innerhalb von Minecraft.

An den Minecraft-Modder Sildur für einen Shader, der sich gut für den Einstieg in die Minecraft-Optifine-Shader-Welt eignet.

References

- BLINN, J. F., AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. *Communications of the ACM* 19, 10, 542–547.
- FILIP, S. 2014. An investigation of fast real-time gpu-based image blur algorithms. URL: <https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms>.
- GLASSNER, A. S. 1989. *An introduction to ray tracing*. Elsevier.
- HOFFMAN, N. 2013. Background: Physics and math of shading. In *SIGGRAPH Courses 2013*. URL: http://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_slides.pdf.
- HOLBERT, D., 2011. Saying 'goodbye' to shadow acne. http://www.dissidentlogic.com/old/images/NormalOffsetShadows/GDC_Poster_NormalOffset.png.
- MARA, M., MCGUIRE, M., NOWROUZEZAHRAI, D., AND LUEBKE, D. 2014. Fast global illumination approximations on deep g-buffers. Tech. Rep. NVR-2014-001, NVIDIA Corporation, June. <http://graphics.cs.williams.edu/papers/DeepGBuffer14/supplemental.pdf>.
- MCGUIRE, M., AND MARA, M. 2014. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques*. URL: <http://jcgta.org/published/0003/04/04/paper.pdf>.
- RUSSEL, J. Basic theory of physically-based rendering. URL: <https://www.marmoset.co/toolbag/learn/pbr-theory>.
- SCHLICK, C. 1994. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, vol. 13, Wiley Online Library, 233–246.
- SOUSA, T., KASYAN, N., AND SCHULZ, N. 2011. Secrets of cryengine 3 graphics technology. In *SIGGRAPH Courses*, ACM, New York, NY, USA. URL: <http://www.crytek.com/cryengine/presentations/secrets-of-cryengine-3-graphics-technology>.
- VAN VLIET, L. J., YOUNG, I. T., AND BECKERS, G. L. 1989. A nonlinear laplace operator as edge detector in noisy images. *Computer Vision, Graphics, and Image Processing* 45, 2, 167–195.

⁴Es besteht bei Minecraft-Optifine-Shadern die Möglichkeit Einstellungen, auf ein Optionen-Menü im Minecraft-Client zu projizieren. Auf diese Weise können Mods oder Shader vom Nutzer nach seinen Wünschen angepasst werden.