Bachelor-Thesis

Vergleich aktueller Screen-Space Ambient Occlusion Verfahren

Vorgelegt von

Rudolf Vocilka

Mat. Nr.: 03916509

3. April 2013

Gutachter: Prof. Dr. Alfred Nischwitz
Betreuer: M.Sc. Andreas Klein

Fakultät für Informatik und Mathematik
Hochschule für angewandte Wissenschaften München

Abstract

Computer graphics display an approximation of real world objects within a computer. One part of computer grafics accounts rendering shadows in real-time. One distincts between direct shadows, that are cast by occluders between a light source and another object within a scene and indirect shadows, that occur due to occluding ambient and reflected light. One technique that calculates the occlusion of ambient light is called Ambient Occlusion (AO). It is not real-time capable.

This thesis covers real-time rendering an approximation of Ambient Occlusion (AO). So far multiple algorithms have been introduced, that calculate an approximation of AO in real-time using the Screen-Space. On the one hand this thesis compares the algorithms Screen-Space Ambient Occlusion (SSAO), Volumetric Obscurance (VO) and Image-Space Horizon-Based Ambient Occlusion (AO) by means of quality and performance. On the other hand we tried to replace the depth-map, that is beeing used to calculate the SSAO algorithms and that is rendered from the cameras perspective, with the depth-map that is used in shadow-mapping and rendered from the lights perspective. In addition we try to improve the visual quality of SSAO, that is is dependent on the selected sample radius. We examine if changing the sample radius may result in a higher quality SSAO.

Comparing the SSAO algorithms yields, that VO and HBAO are much better than SSAO in terms of quality with a comparable performance. VO is more performant, but unflexible because of the neccessary offline sample generation. HBAO is able to vary the number of samples and the resulting quality via shader parameters. Replacing the depth-map in SSAO with the one that is beeing used in shadow mapping introduced new problems, but could work in special situations. This thesis introduces an algorithm for dynamically scaling the SSAO sample radius.

VO may be used for specialized applications that are optimised for perforamance. HBAO is fit for a broad range of hardware by making the sample count a shader parameter. Entirely replacing the depth map results in more problems than the performance gain, of not creating a depth map from the cameras perspective, could justify.

The thesis successfully shows that it is possible to calculate more details by using a dynamic sample radius.

Zusammenfassung

In der Computergrafik werden Objekte der realen Welt vereinfacht im Computer abgebildet. Ein Teilbereich der Computergrafik beschäftigt sich mit der Berechnung von Schatten. Dabei wird zwischen direkten Schatten, die durch lichtundurchlässige Objekte zwischen der Lichtquelle und weiteren Objekten der Szene entstehen und indirekten Schatten, die durch die Verdeckung von ambienten und reflektierten Licht entstehen, unterschieden. Ein nicht echtzeitfähiges Verfahren, mit dem die Verdeckung von ambientem Licht berechnet werden kann, ist Ambient Occlusion (AO).

Diese Arbeit beschäftigt sich mit der echtzeitfähigen Berechnung einer Approximation von AO. Bis dato wurden mehrere Algorithmen vorgestellt, welche die aufwändige Berechnung von AO approximieren, indem die Berechnung im Screen-Space erfolgt. Durch diese Vereinfachung kann AO in Echtzeit berechnet werden. Zum Einen vergleicht diese Arbeit die Algorithmen Screen-Space AO (SSAO), Volumetric Obscurance (VO) und Image-Space Horizon-Based AO (HBAO) anhand der visuellen Qualität und der Performanz. Zum Anderen wurde untersucht, ob die Tiefenkarte aus Sicht der Kamera durch eine Teifenkarte aus Sicht der Lichtquelle, wie sie zur Berechnung beim Shadow-Mapping verwendet wird, ersetzt werden kann. Abschließend soll die visuelle Qualität von SSAO verbessert werden. Bei der Berechnung von AO im Screen-Space hängt die Qualität des Ergebnisses vom gewählten Sample-Radius ab. Es soll untersucht werden, ob die Anpassung des Sample-Radius die Qualität von SSAO steigern kann.

Im Vergleich zeigt sich, dass VO und HBAO qualitativ wesentlich besser als SSAO sind und dabei eine vergleichbare Performanz aufweisen. VO ist dabei performanter, aber wegen der offline Sample-Generierung auch unflexibler. Bei HBAO kann die Sample-Anzahl und damit die resultierende Qualität beliebig durch Parameter gesteuert werden. Die Berechnung von SSAO nur auf Basis der Tiefenkarte aus Sicht der Lichtquelle bringt Probleme mit sich, kann jedoch in speziellen Situationen funktionieren. Diese Arbeit stellt einen neuen Algorithmus vor, durch den der SSAO Sample-Radius dynamisch skaliert werden kann.

VO kann für Anwendungen verwendet werden, die speziell auf Performanz optimiert sind, während die Stärke von HBAO in der flexiblen Konfiguration für verschiedenste Grafikkarten liegt. Des weiteren kann die Tiefenkarte aus Sicht der Kamera nicht komplett durch diejenige aus der Sicht der Lichtquelle ersetzt werden. Die resultierenden Probleme sind unverhältnismäßig hoch, wenn man in Betracht zieht, dass das Erzeugen der Tiefenkarte weniger Zeit in Anspruch nimmt als das Berechnen des SSAO-Wertes.

Es konnte gezeigt werden, dass durch die dynamische Skalierung des Sample-Radius mehr Details einer Szene berechnet werden können.

Diese Erklärung ist zusammen mit der Bachelorarbeit bei der PrüferIn abzugeben.

(Familienname, Vorname)

(Ort, Datum)

(Geburtsdatum)

/ 20
(Studiengruppe / WS/SS)

Erklärung

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

_____
(Unterschrift)

Inhaltsverzeichnis

1 Einleitung

In der Computergrafik werden Objekte der realen Welt näherungsweise im Computer abgebildet. Um beim Betrachter einen dreidimensionalen Eindruck zu erwecken, muss die Beleuchtung der in einer Szene abgebildeten Objekte berechnet werden. Dies kann mit Hilfe des Phong'schen Beleuchtungsmodells, durch die Lichtkomponenten Ambient, Diffus und Spekular, erfolgen.

Dabei werden die Komponenten Diffus und Spekular abhängig von einer Lichtquelle, durch direkte Beleuchtung, berechnet. Das ambiente Licht wird als eine Konstante aufgefasst und so unterstellt, dass die indirekte Beleuchtung eines Objekts durch die Umgebung aus jeder Richtung und in jedem Punkt der Szene gleich ist.

Durch die direkte und indirekte Beleuchtung gibt es auch direkte und indirekte Schatten. Direkte Schatten entstehen durch lichtundurchlässige Objekte zwischen der Lichtquelle und weiteren Objekten der Szene, die ohne ersteres Objekt eigentlich beleuchtet würden. Indirekte Schatten entstehen durch Verdeckung des Umgebungslichts. Da die Beleuchtung von mehreren Objekten abhängt, wird ein globales Beleuchtungsmodell benötigt.

Sobald das Umgebungslicht als variabel aufgefasst wird, reicht ein lokales Beleuchtungsmodell (wie das Phong'sche) nicht mehr aus. Es muss ein globales Modell wie zum Beispiel Radiosity verwendet werden, das wegen der aufwändigen Berechnung zum heutigen Stand nicht für dynamische Szenen echtzeitfähig ist. Die Bachelorarbeit wird sich auf echtzeitfähige Algorithmen beschränken.

Um den Realismus zu verbessern, können zusätzlich zu der lokalen Beleuchtung direkte und indirekte Schatten berechnet werden. Dabei kann Shadow Mapping für direkte und Ambient Occlusion (AO) für indirekte Schatten genutzt werden. AO nimmt zwar ebenfalls ein isotropes Umgebungslicht an, berechnet aber für jeden Punkt im Bild wie viel des Umgebungslichtes verdeckt wird.

Die Berechnung von AO mittels Raytracing ist zu aufwändig, um bei Anwendungen mit Echtzeitanforderungen eingesetzt zu werden. Daher wurden verschiedene Verfahren (wie beispielsweise Screen Space Ambient Occlusion) entwickelt, die die aufwändige Berechnung approximieren. Die Verdeckung des ambienten Lichts der Szene wird also nicht realitätsgetreu berechnet, sondern für den Betrachter glaubhaft angenähert.

Diese Arbeit beschäftigt sich mit verschiedenen echtzeitfähigen Image-Space Algorithmen, namentlich Screen Space Ambient Occlusion (im folgenden SSAO), Image-Space Horizon-Based Ambien Occlusion (im folgenden HBAO) und Volumetric Obscurance (im folgenden VO). Die Algorithmen werden anhand der Performanz und Qualität verglichen. Abbildung 1.1 zeigt ein Bild, dessen rechte Hälfte mit HBAO berechnet wurde.

Des weiteren wurde erfolgreich versucht, die Qualität der SSAO-Verfahren

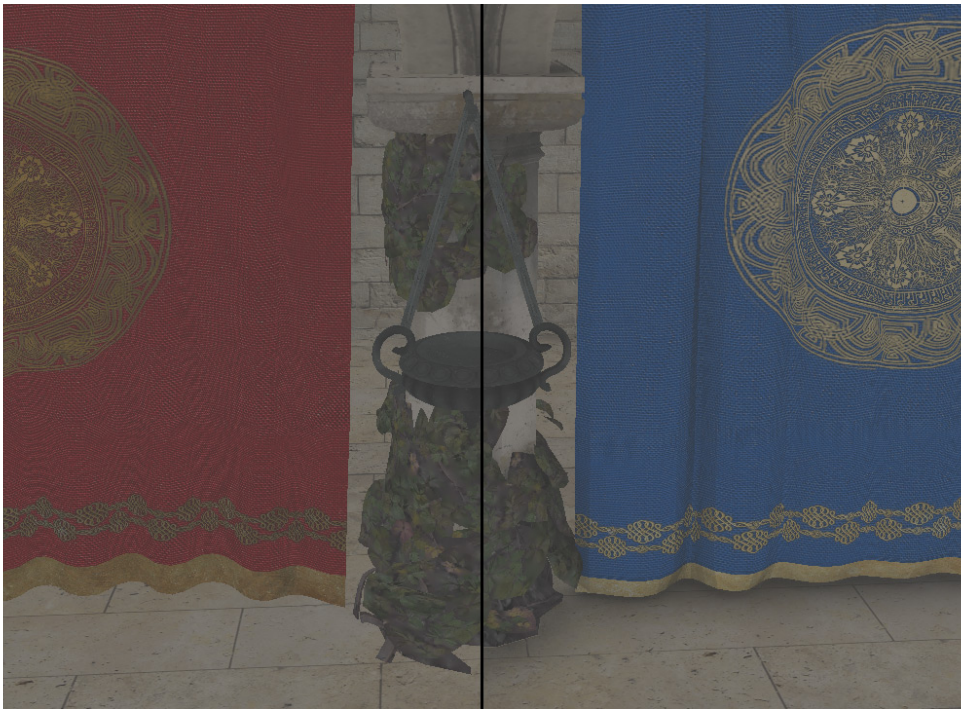
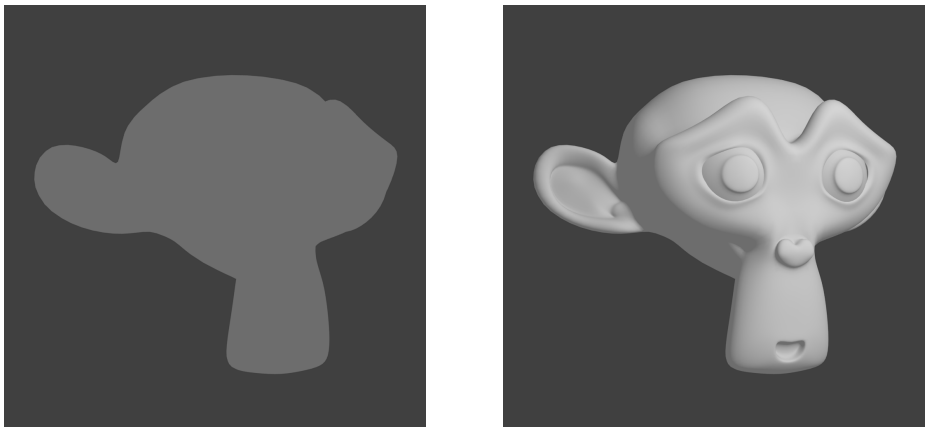


Abbildung 1.1: Der linke Teil der Szene wurde ohne HBAO gerendert, der rechte Teil mit HBAO. Im rechten Bereich ist die Position des Wandteppichs und dessen Falten durch die berechneten Schatten besser zu erkennen.

durch dynamische Skalierung des Radius zu verbessern. Der Radius sollte anhand der Amplitude in der Tiefenkarte so skaliert werden, dass mehr Details sichtbar werden. Ist der Sampleradius an einer Stelle zu groß, dann gehen feine Details verloren, die bei entsprechender Skalierung wieder zum Vorschein gebracht werden können.



(a) Ohne Beleuchtung (b) Mit Beleuchtung

Abbildung 2.1: (a) Durch homogene Beleuchtung erscheint das Objekt flach. (b)Die Form des Objektes wird durch seine Beleuchtung wahrgenommen. Grafik nach Vorbild [NFHS11, Kapitel 12: Beleuchtung und Schattierung]

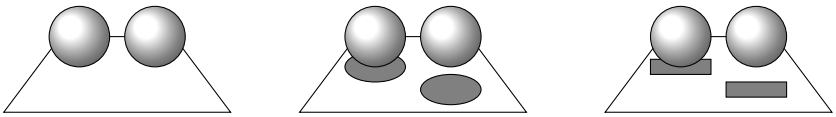
2 Grundlagen

2.1 Menschliche Wahrnehmung

In der Computergrafik werden 3-dimensionale Objekte auf einer 2D Oberfläche, dem Computerbildschirm, dargestellt. Um dem Betrachter einen 3-dimensionalen Eindruck zu vermitteln, muss die Beleuchtung der Oberfläche des Objekts abhängig von einer Lichtquelle berechnet werden. Die Teile der Oberfläche die von der Lichtquelle abgewandt sind, erscheinen dunkler als die Teile die ihr zugewandt sind. Auf diese Weise kann das menschliche Gehirn die Form des Objektes rekonstruieren, was „Formwahrnehmung aus Beleuchtung“genannt wird [NFHS11, Kapitel 12: Beleuchtung und Schattierung].

In Abbildung 2.1 ist das Modell eines Affenkopfes zu sehen. Bei dem Bild 2.1-a wurde durch homogene Beleuchtung jedem Pixel des Kopfes der selbe Farbwert zugewiesen, weshalb das Objekt flach erscheint. Bei Bild 2.1-b kann die Form des Kopfes wahrgenommen werden.

Außerdem ist für die menschliche Wahrnehmung der Szene Schatten von Bedeutung. Durch diesen erhält das Gehirn Informationen über die räumliche Anordnung der Objekte innerhalb der Szene [NFHS11, Kaptel 14: Schatten]. In Abbildung 2.2 wird ersichtlich, dass es nicht ausreichend ist die Objekte lediglich zu beleuchten (engl. shading) (a). Für die korrekte Wahrnehmung der Position innerhalb der Szene wird ein Schatten benötigt (b). Dabei muss es sich nicht zwangsläufig um einen korrekten Schatten handeln (c).



(a) Die Darstellung ohne Schatten erweckt den Anschein, dass sich die Kugeln nebeneinander befinden.

(b) Durch das Hinzufügen eines Schattens erhält die Szene weitere Informationen bezüglich der Position der Kugeln und das menschliche Gehirn interpretiert sie anders. Die rechte Kugel scheint nun über der Ebene zu schweben, während die Linke Kugel sie anscheinend berührt.

(c) Die Form des Schattens ist für die Interpretation der Position für das menschliche Gehirn nicht sonderlich wichtig. Es ist besser einen falsche Schatten zu haben, als keinen.

Abbildung 2.2: Wahrnehmung von Objektpositionen innerhalb einer Szene. Es werden Zwei Kugeln mit (b) Schatten, mit falschem Schatten (c) und ohne Schatten (a) dargestellt. Grafik nach [NFHS11, Kapitel 14: Schatten]

2.2 Beleuchtungsmodelle

Ein Beleuchtungsmodell in der Computergrafik legt fest, wie die Strahldichte (engl. radiance) eines Oberflächenpunktes berechnet wird. Dabei wird in der Computergrafik gegenüber der Radiometrie vereinfacht angenommen, dass die Größen Strahldichte und Strahlenfluss nicht von der Wellenlänge abhängig ist. Die Rendergleichung (2.1) von Kajiya beschreibt, wie die ausgehende Strahldichte L_0 am Punkt p in die Betrachtungsrichtung v berechnet wird [Kaj86].

$$L_0(p,v) = L_e(p,v) + \int_\Omega f(l,v) \otimes L_0\left(r\left(p,l\right), -l\right) \cos\theta \mathrm{d}\omega \tag{2.1}$$

Dabei ist $f(l,v)$ die BRDF (bidirectional reflectance distribution function) ausgewertet für die eingehende Richtung l und der Betrachtungsrichtung v. Laut [AMHH11, S.223-225] wird in der Radiometrie die BRDF genutzt um zu beschreiben, wie Licht von einer Oberfläche in Abhänigkeit von zwei Richtungen (hier l und v) gespiegelt wird. Der Wert der BRDF hängt außerdem von der Wellenlänge ab und wird in der Praxis als ein RGB-Vektor repräsentiert. L_e ist die emittierte Strahldichte von p in Richtung v; L_0 ist die ausgehende Strahldichte von einem anderen Punkt, definiert durch die raycasting Funktion $r(p,l)$[1], in der entgegengesetzten Lichtrichtung $-l$ zum

[1]Die Funktion gibt die Position des ersten Oberflächenpunktes, der mit einem Strahl von p in Richtung l getroffen wird, zurück.

Punkt p. θ ist der Winkel zwischen l und der Oberflächen-Normale beim Punkt p. Es wird über alle $l \in \Omega$ integriert.

Die Gleichung 2.1 besagt, dass die Strahldichte $L_0(p, v)$ sich aus dem emittierten und dem von anderen Punkten reflektierten Licht zusammen setzt. Das reflektierte Licht ist dabei rekursiv definiert. Um ein L_0 zu berechnen muss das reflektierte Licht von anderen Punkten über dieselbe Gleichung gelöst werden. Für die anderen Punkte ist die selbe Gleichung ebenfalls zu lösen usw. Die Rendergleichung summiert alle möglichen Pfade, die ein Lichtstrahl in einer Szene nehmen kann. Sie beschreibt vollständig die Ausbreitung von Licht innerhalb einer Szene und es handelt sich um ein globales Beleuchtungsmodell.

Es wird zwischen lokalen und globalen Beleuchtungsmodellen unterschieden. Transparenz, Reflektionen und Schatten sind laut [AMHH11, S.328] Merkmale für globale Beleuchtungsmodelle, da sie Informationen von anderen Objekten benötigen. Ein lokales Beleuchtungsmodell hat keine Informationen über andere Objekte in der Szene. Bei der Berechnung der Strahldichte wird lediglich die Informationen an der Oberfläche von p benötigt. Die Reflektionsgleichung (2.2), wie sie in [AMHH11, S. 288] vorgestellt wird, ist ein lokales Beleuchtungsmodell.

$$L_0(p, v) = \int_\Omega f(l, v) \otimes L_i(p, l) \cos\theta \mathrm{d}\omega \tag{2.2}$$

Im Vergleich mit der Rendergleichung wurde der Term L_0 mit dem Term $L_i(p, l)$ ersetzt und der Anteil der emittierenden Strahlung L_e entfernt. L_i ist die eingehende Strahldichte an p aus der Richtung l. Zur Berechnung der Reflektionsgleichung werden keine Informationen über andere Objekte in der Szene benötigt. Sie ist nicht rekursiv definiert und lässt sich in Echtzeit berechnen. Bei einem lokalen Beleuchtungsmodell hängt der Wert L_0 also von Materialeigenschaften (BRDF) und dem Winkel zwischen Lichtquelle in Richtung l und Betrachter in Richtung v ab. Ein lokales Beleuchtungsmodell ist in Echtzeit zu berechnen und ermöglicht bereits die Wahrnehmung von Formen, wie sie in Kapitel 2.1 dargestellt wurde.

2.3 Ambientes Licht

Ambientes Licht ist laut [AMHH11, S. 295] das einfachste Modell das indirekte Beleuchtung beschreibt, da die indirekte Strahldichte nicht nach der Richtung variiert und somit einen konstanten Wert L_A hat. Anders ausgedrückt wird die indirekte Beleuchtung als isotrop, also aus jeder Richtung gleich, angenommen. Selbst ein derart einfaches Modell von indirektem Licht verbessert die visuelle Qualität, da eine Szene ganz ohne indirektes Licht unrealistisch wirkt. Objekte im Schatten oder der Lichtquelle abgewandt würden ohne ambientes Licht komplett Schwarz dargestellt.

Die Reflektionsgleichung 2.2 kann um einen Ambient-Term erweitert werden, wie es in Gleichung 2.3 zu sehen ist.

$$L_0(p,v) = c_{\text{amb}} \otimes L_A + \int_\Omega f(l,v) \otimes L_i(p,l) \cos\theta \mathrm{d}\omega \tag{2.3}$$

Dabei ist c_{amb} ein konstanter RGB-Wert der angibt, wie viel des ambienten Lichts L_A (unabhängig von der Blickrichtung v) reflektiert wird. Die Gleichung geht von einer ideal diffusen Reflektion aus. Es ist aber auch möglich die Gleichung 2.3 für allgemeine BRDFs zu definieren, aber in der Praxis wird c_{amb} als konstant angenommen. Für eine Definition mit BRDF sei auf [AMHH11, S. 296] verwiesen.

Die Gleichung 2.3 ignoriert die Tatsache, dass das ambiente Licht nicht aus jeder Richtung an einem Punkt p eintrifft. Einige der Richtungen sind durch Geometrie verdeckt, es könnte eigentlich keine indirekte Beleuchtung aus einer verdeckten Richtungen zu p statt finden. Wenn diese Verdeckung ignoriert wird, dann entsteht beim Betrachter ein flacher Eindruck wie er in 2.1a zu sehen ist. Hier erfolgt die Beleuchtung einzig durch den Ambiente-Term.

Zusammenfassend wird in der Praxis die Renderinggleichung vereinfacht, um die Beleuchtung einer Szene in Echtzeit zu berechnen. Die Rekursion (gegeben durch den Term $L_0(r(p,l), -l)$) wurde entfernt und durch eine Lichtquelle $L_i(p,l)$ ersetzt. Durch die Cousinsgewichtung nach dem Winkel θ werden Oberflächen, die von der Lichtquelle abgewandt sind, nicht beleuchtet. Dies wirkt unrealistisch, weshalb der Ambiente-Term hinzugenommen wurde. Um die visuelle Qualität zu verbessern, kann die Verdeckung des ambienten Lichts für den Punkt p berechnet werden.

2.4 Verdeckung von ambienten Licht

2.4.1 Obscurance

Obscurance ist ein empirisches Beleuchtungsmodell für Umgebungslicht und wurde 1998 von Zhukov et al. vorgestellt. Es beruht auf der Beobachtung, dass je weiter ein Objekt von einem Punkt entfernt ist, desto weniger trägt dieses Objekt zu dessen Verdeckung bei.

Es ist definiert als:

$$L_A(P) = \frac{1}{\pi} \int_\Omega \underbrace{p\left(\overbrace{d(P,\omega)}^{\text{Distanz zum Schnittpunkt}}\right)}_{\text{Abschwächung}} \overbrace{\cos\theta}^{\text{Cosinus-Gewichtung}} \mathrm{d}\omega \tag{2.4}$$

Wobei Ω die Hemisphäre, p eine Abschwächungsfunktion, d die Distanz zum ersten Schnittpunkt mit Objekten der Szene und θ der Winkel zischen

der Normalen am Punkt P und ω ist. Durch den Faktor $\cos\theta$ werden Lichtstrahlen in Richtung des Normalenvektors stärker gewichtet. p ist eine empirische Mappingfunktion, die die Distanz zum Schnittpunkt $d(P,\omega)$ auf die Lichtintensität aus der Richtung ω zum Punkt P abbildet.

Je weiter der Schnittpunkt von P entfernt ist, desto mehr Licht kommt dort an. Die Funktion ist monoton steigend, nach oben begrenzt und kann Werte zwischen $[0,1]$ annehmen. Ab einer definierten Distanz wird für größere Distanzen der Wert 1 angenommen, was die Beobachtung abbildet, dass die Helligkeit des Punktes P primär von seiner näheren Umgebung beeinflusst wird [ZIK98].

2.4.2 Ambient Occlusion

Ambient Occlusion kann als ein Spezialfall von Obscurance angesehen werden, bei dem die Abschwächungsfunktion d wie folgt definiert ist [LS10]:

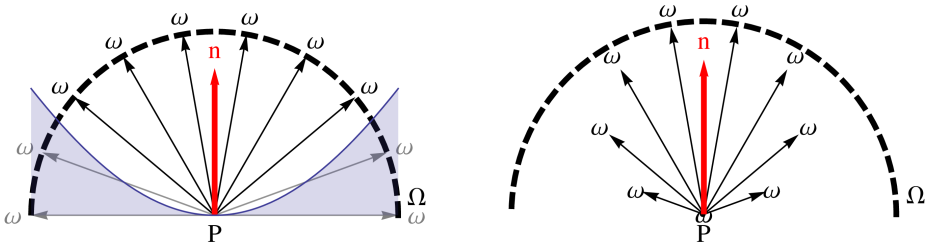
$$p(L) = \begin{cases} 0 & : L < \infty \\ 1 & : sonst \end{cases}$$

Der Begriff Ambient Occlusion wurde 2002 von [Lan02] vorgestellt und von [Chr03] näher definiert. [ZIK98] stellten bereits 1998 ein ähnliches Beleuchtungsmodell vor. Ambient Occlusion ist ein globales Beleuchtungsmodell und approximiert, wie viel des ambienten Lichts an einem Punkt in der Szene durch umliegende Geometrie verdeckt wird. Es approximiert indirekte Schatten, wobei angeommen wird, dass das ambiente Licht isotrop, also aus jeder Richtung gleich, ist.

$$L_A(P) = 1 - \frac{1}{\pi} \int_{\omega \in \Omega} \overbrace{V(P,\omega)}^{\text{Sichtbarkeitstest}} \underbrace{(n \cdot \omega)}_{\text{Cosinus-Gewichtung}} \, \mathrm{d}\omega \tag{2.5}$$

Formel 2.5 zeigt wie AO definiert ist. Der Faktor $\frac{1}{\pi}$ normiert das Ergebnis des Integrals in den Wertebereich $[0,1]$. $V(P,\omega)$ ist eine Sichtbarkeitsfunktion die 1 liefert, wenn ein Strahl von P in Richtung ω Geometrie schneidet und ansonsten 0. Durch die Cosinus-Gewichtung $n \cdot \omega$ werden Vektoren in Richtung der Normalen stärker gewichtet als solche die Senkrecht zur Normalen stehen. Wenn n und ω normiert sind, dann ist $n \cdot \omega$ äquivalent zu $\cos\theta$, wie es bei Obscurance definiert wurde. Abbildung 2.3 zeigt die schematische Darstellung dieser Formel.

Ambient Occlusion vereinfacht gegenüber der Rendergleichung. Durch den Sichtbarkeitstest wird nur entschieden, ob ein Strahl blockiert oder frei ist. In der Rendergleichung kann aus einer für AO blockierten Richtung aber durchaus Licht durch Reflektion zu dem Punkt P gelangen. Obscurance versucht dies durch die Verwendung einer Abschwächungsfunktion zu berücksichtigen, was aber auch eine Vereinfachung darstellt.



(a) Ausrichtung der Hemisphäre um n am Punkt P. Strahlen von P zur Oberfläche der Hemisphäre Ω. Die Geometrie ist in Blau dargestellt. Graue Strahlen bedeuten $V(P,\omega) = 0$, bei schwarzen Strahlen ist $V(P,\omega) = 1$.

(b) Darstellung der Gewichtung als Vektorlänge. Die Vektorlänge entspricht $n \cdot \omega$. Je kleiner der Winkel zwischen ω und n ist, desto mehr wird der Strahl gewichtet.

Abbildung 2.3: Die Ambient Occlusion Hemisphäre

Zusammenfassend approximiert Ambient Occlusion über Raytracing wie viel der indirekten Beleuchtung durch Geometrie verdeckt wird. Je mehr Strahlen Geometrie schneiden, desto mehr des isotropen Umgebungslichtes wird verdeckt. Dabei zählt Geometrie, die sich in Richtung der Normalen befinden mehr als solche, die sich auf der Senkrechten befinden.

3 Stand der Technik

3.1 Screen Space Ambient Occlusion

Das Screen Space Ambient Occlusion Verfahren der Firma Crytek wurde erstmals auf der SIGGRAPH-Konferenz 2007 vorgestellt [Mit07] und in dem Artikel [Kaj09] der genaue Algorithmus beschrieben. Es beruht auf der Idee, die Tiefenkarte zu verwenden um eine Näherung an Ambient Occlusion zu berechnen. Die Berechnung im Screen-Space bietet den Vorteil, dass sie unabhängig von der Komplexität der Szene ist.

3.1.1 Sampling

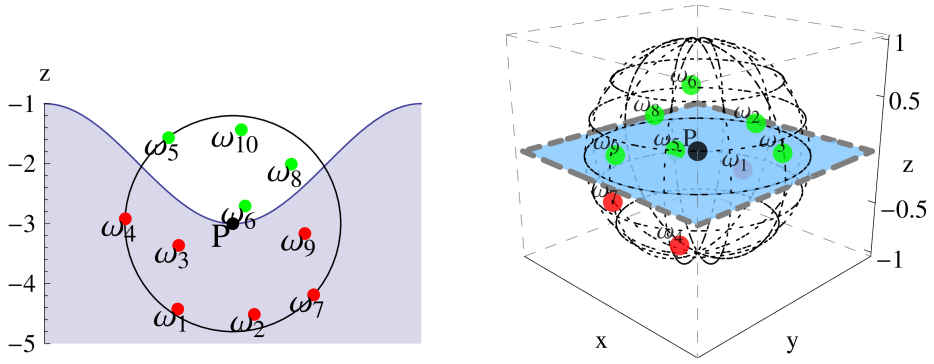
Das Ziel von SSAO ist es, für jeden gezeigten Punkt P der Szene einen Verdeckungsfaktor zu berechnen. Hierbei ist die genaue Beschaffenheit der Szene nicht bekannt, da die (perspektivische) Projektion der Punkte auf eine planare Fläche bereits stattgefunden hat. Es wird eine Tiefenkarte zur Berechnung herangezogen. Die Tiefenkarte enthält für jeden Pixel die Information, wie weit die an dieser Stelle des Screen-Space gezeigte Geometrie von der Kamera entfernt ist. Würde man wie bei Ambient Occlusion Raytracing verwenden um den Term zu berechnen, müssten pro Strahl mehrere Texturzugriffe auf die Tiefenkarte erfolgen. Da dies zum damaligen Zeitpunkt zu aufwendig war um echtzeitfähig zu sein, musste Crytek eine andere Lösung finden um den Wert zu approximieren. Anstatt wie beim Raytracing zu zählen, wie viele Strahlen in der Pixelumgebung auf Geometrie treffen, wird mittels Sampling approximiert, wie viel Prozent der Umgebung des Punktes verdeckt sind. [Kaj09]

Um den Wert zu approximieren wird eine Sphäre um den Punkt P gelegt in der Samples verteilt werden. Abbildung 3.1b zeigt diese Sphäre in räumlicher Darstellung. Für jedes Sample wird der z-Wert des Samples mit dem korrespondierenden z-Wert der Szene anhand der Formel 3.1 verglichen. Dabei sind z_p und z_s lineare Tiefenwerte $\in [0, 1]$, wobei 0 bedeutet, dass der Punkt auf der Near-Plane liegt. Der Wert 1 bedeutet, der Punkt liegt auf der Far-Plane.

$$visibility(z_p, z_s) := \begin{cases} 1 & z_s > z_p \\ 0 & \text{sonst} \end{cases} \tag{3.1}$$

Ist der Tiefenwert des Samples kleiner als der von P, dann ist es von einem Objekt verdeckt. Das Verhältnis von verdeckten zu freien Samples approximiert, wie viel Prozent der Sphäre verdeckt sind.

Abbildung 3.1a zeigt die in einer Sphäre zufällig verteilten Samples ω_1 bis ω_{10} im Querschnitt. Der Verdeckungsfaktor beträgt in diesem Beispiel $\frac{6}{10}$, da 6 der 10 Samples verdeckt sind. Anhand des Beispiels sieht man auch, dass bei einer ebenen Fläche im Durchschnitt 50% der Samples verdeckt sind. Mittring [Mit07] sieht dies als Vorteil, da deswegen Ecken hervorge-



(a) Schematische Darstellung einer Szene. Die horizontale Achse zeigt die Dimensionen x und y, die vertikale die Dimension z. Die Sphäre um den Punkt P ist als Kreis eingezeichnet.

(b) 3D Ansicht der Sphäre um den Punkt P

Abbildung 3.1: SSAO Sampling. Die in der Szene enthaltene Geometrie ist farblich dargestellt. Die grünen Samples sind frei, die roten verdeckt.

hoben werden und weitere Details der Geometrie erkennbar werden. Der Algorithmus weicht hier von der Definition von AO ab.

Der Punkt P befindet sich im Screen-Space auf der Oberfläche des an dieser Stelle gezeigten Objektes. Für ihn hat bereits die Projektion auf eine Fläche stattgefunden. Für ihn ist also die Position auf dem Bildschirm und der zugehörige z-Wert bekannt. Mit Hilfe dieser Informationen kann die View-Space Position wiederhergestellt werden.

Bei der Implementierung kann entweder auf einer Scheibe im Screen-Space oder innerhalb der Sphäre im View-Space gesamplet werden. Die Sphäre im View-Space ist nach der Projektion eine Scheibe im Screen-Space. Die Samples auf der Scheibe im Screen-Space können wieder in den View-Space zurück gerechnet werden.

3.1.2 Sample Invalidierung

Der Sichtbarkeitstest aus Formel 3.1 berücksichtigt nicht, wie weit ein Sample vom Mittelpunkt P der Sphäre entfernt ist. Um zu verhindern, dass weit entfernte Geometrie einen Schatten auf Punkt P werfen, müssen Samples als ungültig erklärt werden. Würden weit entfernte Samples nicht als ungültig erklärt, könnte es beispielsweise vorkommen, dass eine Säule einen Schatten auf eine Wand wirf die zu weit entfernt ist um von der Säule beeinflusst zu werden.

Laut Kajalin [Kaj09] ist die Sample-Validität gegeben durch die Formel 3.2. Diese erwartet zwei lineare Tiefenwerte $\in [0, 1]$: z_p ist die Tiefe von P und z_s die Tiefe des Samples.

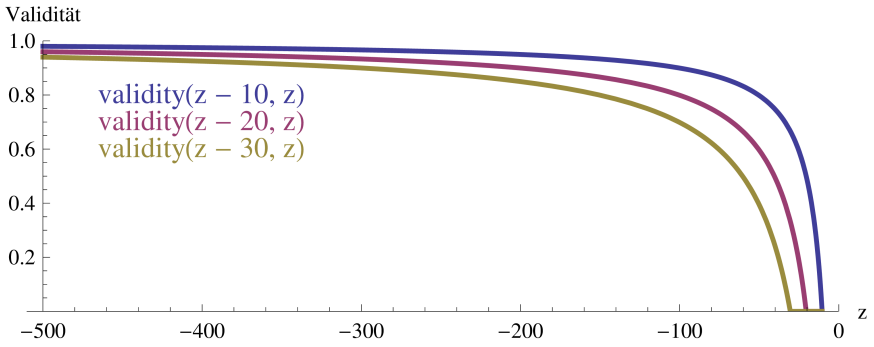


Abbildung 3.2: Validität von Sample-Differenzen in Abhängigkeit der Tiefe des Sample-Punktes z_p. Die horizontale Achse zeigt die Viewspace z-Werte, die vertikale die Validität der Differenz. Das Beispiel geht von einer Near-Plane bei 0,5 und einer Far-Plane bei 500 aus. Der Plot der Funktion zeigt, dass die Sample-Invalidierung abhängig von der Entfernung zur Kamera ist. Eine Differenz von 10, 20 bzw. 30 Viewspace Einheiten von z_s zu z_p ist nahe an der Kamera (z_p geht gegen -0,5) weniger valide als bei größerer Entfernung (z_p geht gegen -500)

$$validity(z_p, z_s) := 1 - \max\left(0, \min\left(1, \frac{z_p - z_s}{z_s}\right)\right) \tag{3.2}$$

Der Sichtbarkeitswert aus Formel 3.1 wird mit der Validität zwischen den Werten 0,5 und *visibility* linear interpoliert um den endgültigen Sample-Wert zu berechnen. Ist $validity(z_p, z_s)$ gleich 0, dann ist der Sample-Wert 0,5. Ist *validity* gleich 1, dann ist der Sample-Wert gleich *visibility*. Mathematisch ausgedrückt:

$$\begin{aligned} mix(x, y, a) &:= x \cdot (1 - a) + y \cdot a \\ sampleValue(z_p, z_s) &:= mix(0.5, visibility(z_p, z_s), validity(z_p, z_s)) \end{aligned} \tag{3.3}$$

Durch die Sample-Invalidierung wird erreicht, dass Geometrie Lichtstrahlen nur innerhalb einer bestimmten Distanz verdecken kann. Bei Obscurance ist dieser Umstand durch die Abschwächungsfunktion p aus der Gleichung 2.4 gegeben. Bei Ambient Occlusion werden die Strahlen zwar nicht nach Distanz abgeschwächt, aber in der praktischen Implementierung werden sie nur bis zu einer gewissen Distanz verfolgt. Diese Distanz ist bei Obscurance und Ambient Occlusion im Viewspace gegeben. Bei SSAO wird diese Distanz jedoch relativ zur Sample-Tiefe z_s gewertet. Wie in Abbildung 3.2 zu sehen ist, hängt die Validität einer Sample-Differenz daher von der Entfernung zur Kamera ab.

(a) Ohne gespiegelte Samples

(b) Zufällig gespiegelte Samples. Mehr Richtungen werden abgetastet, aber es entsteht Rauschen.

Abbildung 3.3: SSAO mit und ohne Spiegelung der Samples

3.1.3 Erzeugen von mehr Samples durch Spiegeln

Durch die Echtzeitanforderung können nur eine begrenzte Anzahl an Samples erhoben werden. Durch die geringe Anzahl an Samples (engl. Undersampling) entstehen hochfrequente, harte Schatten, da immer mit den gleichen Samplepositionen abgetastet wird. Diese Artefakte werden auch Banding-Artefakte genannt. Dies ist in Abbildung 3.3a zu sehen. Um die Anzahl an Sample-Richtungen bei geringer Anzahl zu erhöhen, kann der Sample-Kernel pseudozufällig rotiert werden [Isi13]. Anstatt den Kernel zu rotieren kann jedes Sample gespiegelt werden, um die Performance zu erhöhen [Mit07]. Durch das zufällige Spiegeln der Samples werden nun mehr Richtungen abgetastet, es entsteht jedoch ein hochfrequentes Rauschen. Beides ist in Abbildung 3.3b zu sehen.

Das Rauschen kann durch einen Tiefpassfilter entfernt werden, der nicht über Kanten hinweg glättet [Kaj09].

3.2 Volumetric Obscurance

Volumetric Obscurance steht in Relation zu dem in Kapitel 2.4.1 vorgestellten Obscurance und wurde 2010 von Loos und Sloan [LS10] vorgestellt. Die Berechnung des Oclussion-Werts erfolgt bei diesem Verfahren nicht über Raytracing, wie es bei Obscurance der Fall ist, sondern über das sogenannte Line-Sampling der Tiefenkarte. Formel 3.4 zeigt wie VO definiert ist.

$$V(P) = \int_X \overbrace{p\left(\underbrace{d(P,x)}_{\text{Distanz}}\right)}^{\text{Abschwächungsfunktion}} \underbrace{O(x)}_{\textit{Sichtbarkeitsfunktion}} \mathrm{d}x \tag{3.4}$$

Dabei sind X benachbarte Punkte um P und $O(x)$ eine Sichtbarkeitsfunktion die gleich 0 ist, wenn sich an x ein Objekt befindet und ansonsten den Wert 1 annimmt. p ist eine Abschwächungsfunktion die beim Punkt P gleich 1 ist und bei einer festgelegten Distanz den Wert 0 annimmt. Die Autoren haben eine quadratische mit einer konstanten Abschwächungsfunktion verglichen und stellten fest, dass die Unterschiede so subtil sind, dass es nicht die zusätzlichen Berechnungskosten rechtfertigt. Daher wurde die Funktion p als konstant definiert [LS10].

3.2.1 Line-Sampling

Die Berechnung von Volumetric Obscurance erfolgt nicht ausschließlich numerisch. Die Tiefeninformation wird analytisch abgehandelt und die anderen beiden Dimensionen numerisch [LS10]. Ein Sample wird als eine Linie angesehen, die parallel zur z-Achse ist und mit der Funktion f in Formel 3.5 definiert wird. Dabei ist z_p der Tiefenwert des Punktes P und d der Tiefenwert des Samples. Die Line-Samples sind in Abbildung 3.4a zu sehen.

Alle Line-Samples werden auf einer Einheitsscheibe verteilt, die in Abbildung 3.4b zu sehen ist. Ein Line-Sample befindet sich auf der Position (x, y) Einheiten vom Mittelpunkt der Scheibe entfernt.

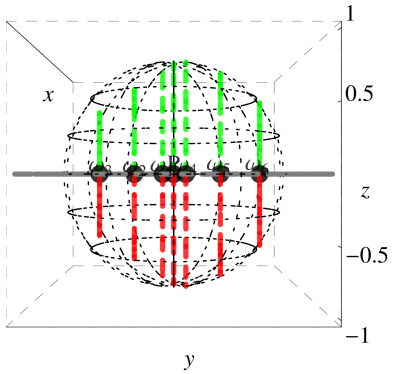
$$f(z_p) = \begin{cases} 1 & : z_p \leq d \\ 0 & : z_p > d \end{cases} \tag{3.5}$$

Um zu berechnen wie viel des Line-Samples verdeckt ist, wird das Intergral aus Formel 3.6 ausgewertet. Dabei ist $z_s = \sqrt{1 - x^2 - y^2}$ die Länge des Line-Samples abhängig von dessen Position auf der Einheitsscheibe. Damit ist ein Line-Sample innerhalb einer Einheitssphäre mit dem Radius 1 definiert. Die Einheitssphäre ist in Abbildung 3.4c zu sehen.

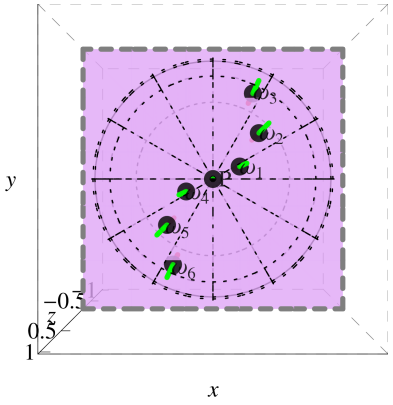
$$\int_{-z_s}^{z_s} f(z)dz = \max\left(\min\left(z_s, d_r\right) + z_s, 0\right) \tag{3.6}$$

3.2.2 Samplevolumen

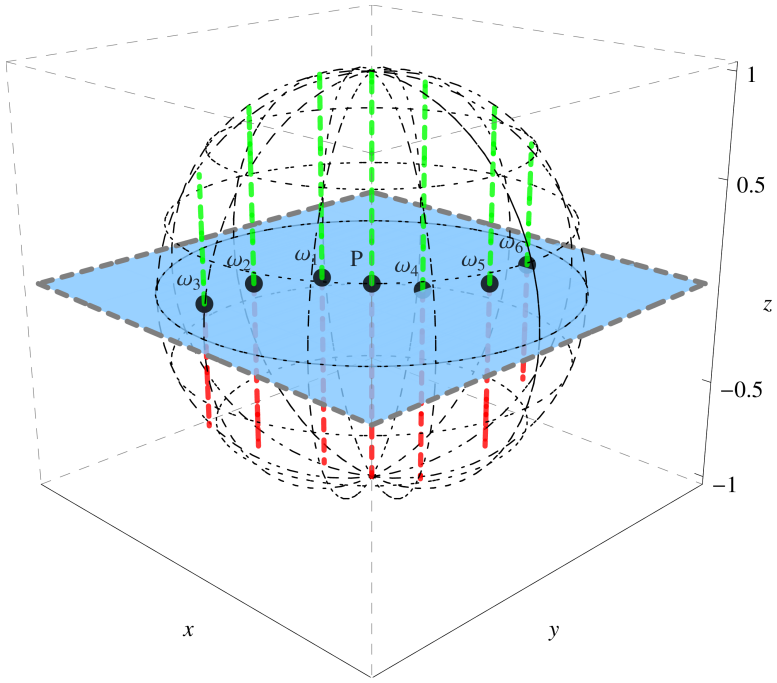
Durch die analytische Auswertung des Integrals aus Formel 3.6 ist bekannt, wie viel Prozent jedes Line-Samples verdeckt sind. Um zu berechnen, wie viel des Volumens der Einheitssphäre verdeckt ist, wird jedem einzelnen Line-Sample ein Volumen zugeordnet. Das Volumen einer Sample-Position wird über ein Voronoi-Diagramm, zu sehen in Abbildung 3.5, ermittelt. Jedem Sample wird eine Farbe zugeordnet und die Positionen um das Sample innerhalb des Einheitskreises ausgewertet. Eine Position erhält die Farbe eines Samples, wenn es kein anderes Sample gibt das näher an dieser Position ist. Die Summe über alle Positionen, die einem Sample zugeordnet sind, multipliziert mit dem z_s-Wert entspricht dem Volumen des Samples. Loos und



(a) In der Seitenansicht sind die Line-Samples zu erkennen. Rot bedeutet verdeckt, grün frei. Die Fläche ist als graue Linie eingezeichnet.

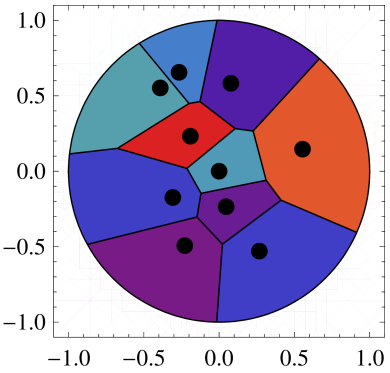


(b) Aus der Kameraperspektive (Blickrichtung $-z$) ist die Einheitsscheibe zu sehen, auf der die Samples verteilt werden. Die Fläche ist in Rosa eingezeichnet. Diese Ansicht gleicht dem Blick auf die Tiefenkarte.

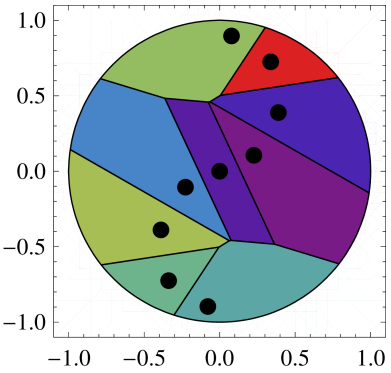


(c) Isometrische Ansicht der Einheitssphäre. Die Fläche ist in blau eingezeichnet.

Abbildung 3.4: Line-Samples P, $\omega_1, \omega_2, ...\omega_6$ dargestellt in der Einheitssphäre, deren Sample-Punkt P sich auf einer planaren Fläche befindet. Die Fläche ist parallel zum Betrachter ausgerichtet und die Line-Samples sind jeweils zu 50% verdeckt.



(a) Zufällig verteilte Samples



(b) Werden alle Samples über den Ursprung gespiegelt, dann ist der Bereich der gespiegelten Samples so groß wie der des Pendants.

Abbildung 3.5: Voronoi-Diagramm für Line-Samplepositionen auf der Einheitsscheibe

Sloan haben die Samplepositionen auf der Einheitsscheibe so verteilt, dass alle Samples das gleiche Volumen erhalten.

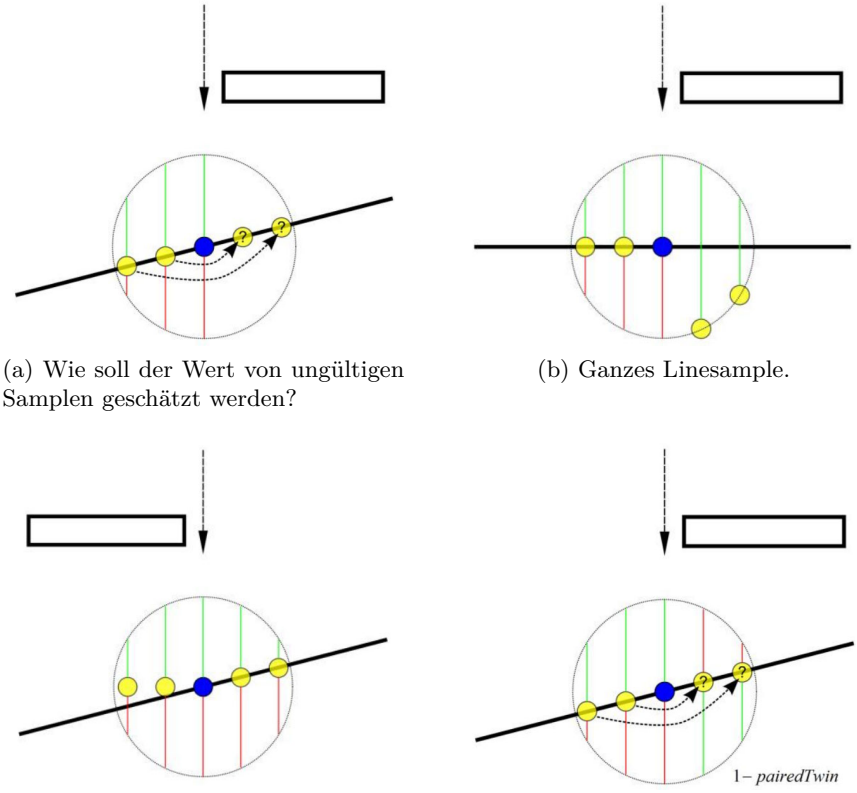
3.2.3 Sample Invalidierung

Wie bei SSAO müssen auch bei VO Samples invalidiert werden, wenn die Tiefenwert des Samples zu weit von dem des mittleren Punktes P entfernt ist. Ownby [JPO10] beschreibt die Verwendung von Sample-Paaren. Wird ein Sample für ungültig erklärt, so kann durch das verwenden des Umkehrwerts des Sample-Paares eine ebene Fläche simuliert werden. Abbildung 3.7a zeigt, wie ein Sample-Paar durch spiegeln eines Sample-Punktes über den Ursprung erzeugt wird.

Wird ein Sample für ungültig erklärt, dann muss ein Ersatzwert für dieses gefunden werden. Würde man, wie in Abbildung 3.6b gezeigt, annehmen, dass das Sample komplett unverdeckt ist, dann würden an diesen Stellen helle Bereiche entstehen. Würde man hingegen annehmen, dass ein Sample zu 50% frei ist, würde bei Abbildung 3.6c ein leichter Schatten entstehen. Die stabilsten Ergebnisse erhält man laut [JPO10] durch das Verwenden von Sample-Paaren wie es in Abbildung 3.6d gezeigt ist.

3.2.4 Erzeugen von mehr Samples durch Rotation

Wie bei SSAO besteht auch bei VO das Problem, dass zu wenige Samples verwendet werden. Daher müssen auch bei VO neue Sample Konstellationen erzeugt werden. Im Unterschied zu SSAO sind die Samples aber im Zweidimensionalen verteilt und es werden Sample-Paare verwendet. Diese sind auf



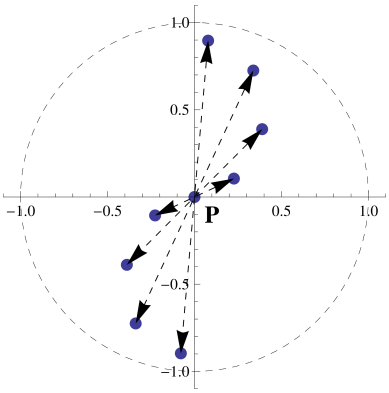
(a) Wie soll der Wert von ungültigen Samplen geschätzt werden?

(b) Ganzes Linesample.

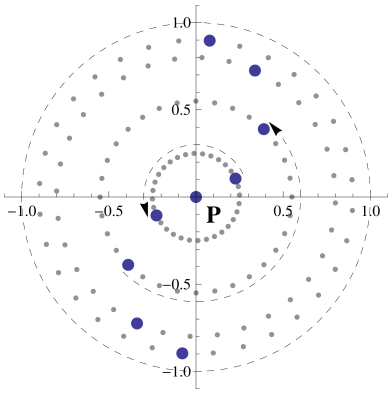
(c) Halbes Linesample.

(d) Umkehrwert des Samplepaares.

Abbildung 3.6: Verschiedene Möglichkeiten einen Wert für ein ungültiges Sample zu schätzen. 3.6b erzeugt einen hellen Rand, 3.6c erzeugt einen leichten Schatten, was durch das ungültig erklären eigentlich verhindert werden soll. Durch 3.6d kann eine Ebene Fläche angenommen werden. Grafiken [JPO10]

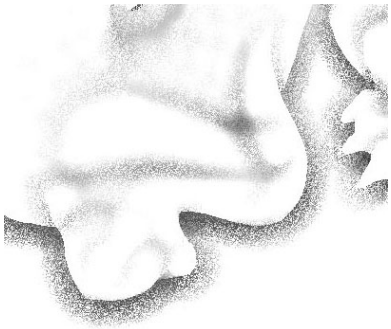


(a) Durch Spiegeln über den Ursprung werden Samplepaare erzeugt



(b) Durch das benutzen von Paired-Samples muss nur um 180° gedreht werden

(c) Ohne Rotation



(d) Mit Rotation

Abbildung 3.7: Samples dargestellt auf der Einheitsscheibe. Die Betrachtungsweise entspricht der aus Abbildung 3.4b. Darunter VO mit und ohne Rotation der Samples

einer Scheibe, wie in Abbildung 3.7a dargestellt, verteilt. Da die Paare durch spiegeln über den Ursprung erstellt werden reicht es aus, die Sample-Punkte um 180° zu drehen, um den vollen Kreis abzudecken (Abbildung 3.7b). Abbildung 3.7c zeigt VO ohne Rotation. Es ist zu sehen, dass nicht in jede Richtung abgetastet wird und so der horizontale Schattenverlauf fehlt. Mit Rotation 3.7d wird jede Richtung beachtet, es entsteht jedoch ein Rauschen. Das Rauschen kann durch einen Tiefpassfilter entfernt werden.

3.3 Image-Space Horizon-Based Ambient Occlusion

Image-Space Horizon-Based Ambient Occlusion wurde 2008 auf der SIGGRAPH-Konferenz vorgestellt und in dem Artikel [BSD09] beschrieben. Es beruht

auf der Idee durch sogenanntes Ray marching, also durch Samplen entlang einer Linie in der Tiefenkarte, von einem Punkt P in eine bestimmte Richtung einen Horizont zu ermitteln. Der Horizont entspricht dem höchsten Punkt in der Tiefenkarte, der in eine Richtung gefunden wurde.

Die Autoren definieren die Ambient Occlusion Gleichung 2.5 im Sphärenkoordinatensystem, wie es in Gleichung 3.7 zu sehen ist. Die Sphäre ist nach der z-Achse ausgerichtet. Dabei ist $W(\vec{\omega})$ eine Abschwächungsfunktion, $V(\vec{\omega})$ ist eine Sichtbarkeitsfunktion die 1 liefert, wenn ein Strahl in Richtung ω Geometrie schneidet und ansonsten gleich 0 ist. ϕ ist der Azimutwinkel und θ der Polarwinkel. $t(\theta)$ ist der Winkel der Tangentenfläche an dem Punkt P, die senkrecht zur Normalen an diesem Punkt ist. Die Gleichung 3.7 ist in Abbildung 3.8a dargestellt.

$$A = 1 - \frac{1}{2\pi} \int_{\theta=-\pi}^{\pi} \int_{\phi=t(\theta)}^{t(\theta)+\frac{\pi}{2}} V\left(\vec{\omega}\right) W\left(\vec{\omega}\right) \mathrm{d}\omega \tag{3.7}$$

Es wird angenommen, dass die Nachbarschaft um P im Sample-Radius R in der Tiefenkarte kontinuierlich ist. Zu diesem Zweck werden alle Samples die außerhalb des Sample-Radius liegen als ungültig erklärt. Da die Tiefenkarte kontinuierlich ist, treffen alle Strahlen mit dem Azimutwinkel ϕ zwischen der Tangentenfläche $t(\theta)$ und dem Horizont $h(\theta)$ auf Geometrie ($V(\vec{\omega}) = 1$). Die Gleichung 3.7 lässt sich dann wie in Gleichung 3.8 definieren. Dabei ist $\phi = h(\theta)$ der Azimutwinkel zum Horizont in Richtung θ. Der Abschwächungsfunktion W wird die Distanz zum Horizont übergeben. Der Horizontwinkel ist in Abbildung 3.8b dargestellt.

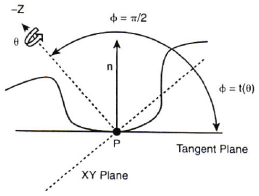
$$A = 1 - \frac{1}{2\pi} \int_{\theta=-\pi}^{\pi} \int_{\phi=t(\theta)}^{h(\theta)} W\left(\vec{\omega}\right) \cos\left(\phi\right) \mathrm{d}\phi \mathrm{d}\theta \tag{3.8}$$

Zum Lösen der Gleichung 3.8 müssen die beiden Integrale ausgewertet werden. Das äußere Integral wird in X Richtungen ausgewertet. Das innere Integral, also der Horizontwinkel, wird durch Ray-Marching in Y Schritten gelöst. Hierzu erfolgen Texturzugriffe in die Tiefenkarte um den Punkt P in eine Richtung θ innerhalb des Sample-Radius in Y Schritten. Die Anzahl an Richtungen und Schritten sind als Parameter an HBAO gegeben.

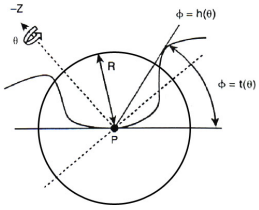
Um zu verhindern, dass Occlusion durch niedrige Tesselierung von Geometrie entsteht, haben die Autoren einen Angle Bias definiert. Ein Horizontwinkel, der kleiner als der Angle Bias ist, wird nicht gewertet.

3.4 Grenzen des Screen-Space Ambient Occlusion

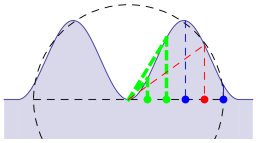
Der Vorteil der Berechnung im Screen-Space, die unabhängig von der Komplexität der Szene ist, bringt aber auch Probleme mit sich, die bei von Kajalin [Kaj09] beschrieben wurden. Zum Einen fehlt im Screen-Space Geometrie



(a) Parameter der nach Normale n ausgerichteten Hemisphäre um P. Grafik [BSD09]



(b) Horizontwinkel $h(\theta)$. Grafik [BSD09]

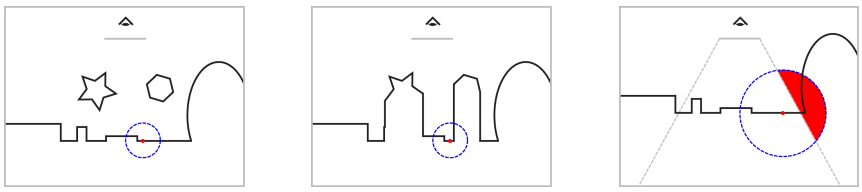


(c) Ray marching von der Mitte des Bildes nach rachts. Die grünen Linien bedeuten, dass der Horizontwinkel vergrößert wurde und das Sample (als Punkt eingezeichnet) akzeptiert wurde. Rot bedeutet der Horizontwinkel wurde kleiner. Blaue Punkte liegen außerhalb des Radius.

Abbildung 3.8: Grafiken zu HBAO mit der Hemisphäre und dem Horizontwinkel und Ray marching

die beim Rendern außerhalb des View-Frustums lag (Abbildung 3.9c). Bavoil und Sainz schlagen als Lösung ein vergrößertes View-Frustum vor [LB13a]. Zum Anderen sind die Objekte in der Tiefenkarte unendlich dick (Abbildung 3.9b). Dadurch kann Geometrie, die aus Sicht der Kamera durch andere Geometrie verdeckt ist, nicht zur Occlusion-Berechnung herangezogen werden, da diese Informationen fehlen. Ohne geeignete Sample-Invalidierung führen Situation wie in Abbildung 3.9b zur Überbewertung des Occlusion-Werts im Vergleich zur Berechnung im Viewspace (Abbildung 3.9a). Diese Probleme treten bei allen Screen-Space Ambient Occlusion Algorithmen auf, die auf einer einzigen Tiefenkarte beruhen. Bavoil und Sainz stellten die Verwendung von mehreren Schichten von Teifenkarten vor, um dieses Problem zu beseitigen [LB13a]. Vardis et al. kombinieren mehrere Tiefenkarten aus verschiedenen Perspektiven, um an fehlende Informationen zu gelangen [VPG13].

Außerdem hängt die Qualität des SSAO auch von dem gewählten Sample-Radius ab. In Abbildung 3.10 werden die Screen-Space Algorithmen mit AO verglichen. Sowohl SSAO in Abbildung 3.10b als auch VO in Abbildung 3.10c bewerten die Occlusion mit dem gezeigten Sample-Radius als zu klein. Einzig HBAO (Abbildung 3.10d) liefert bei diesem Radius ein zu AO in Abbildung 3.10a vergleichbares Ergebnis. Wird der Sample-Radius verkleinert, dann bewerten SSAO (Abbildung 3.10f) und VO (Abbildung 3.10g) die Situation ebenfalls vergleichbar zu AO (Abbildung 3.10e).

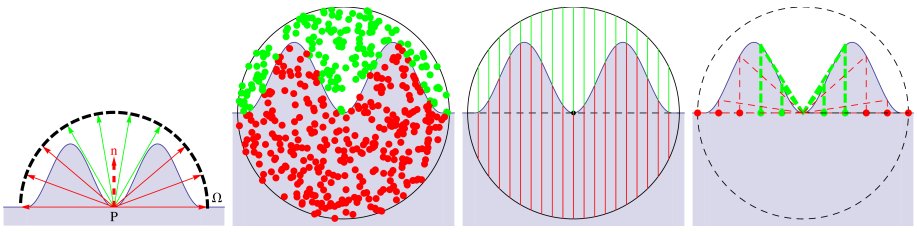


(a) Im Viewspace ist der rot eingezeichnete Punkt kaum verdeckt

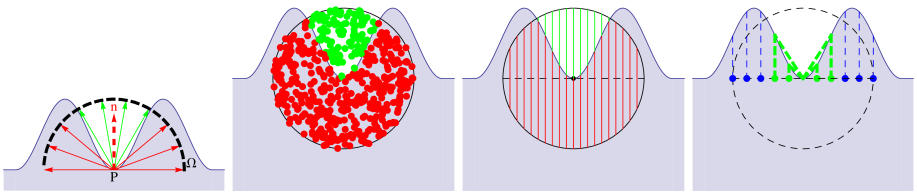
(b) Im Screen-Space ist der rot eingezeichnete Punkt stark verdeckt

(c) Geometrie außerhalb des View-Frustums trägt nicht zur Verdeckung bei

Abbildung 3.9: Probleme bei der Berechung im Screen-Space. Geometrie in der Tiefenkarte ist unendlich dick (3.9b) und es sind nicht alle für die Occlusion-Berchnung relevanten Informationen vorhanden (3.9c). Grafiken [LS10]



(a) AO (b) SSAO (c) VO (d) HBAO



(e) AO (f) SSAO (g) VO (h) HBAO

Abbildung 3.10: Vergleich der Screen-Space Algorithmen mit Ambient Occlusion. Die grünen Bereiche werden als frei bewertet. Bei HBAO ist der Bereich zwischen den Horizonten, hier durch grüne Linien dargestellt, frei. Die Abbildungen 3.10a bis 3.10d in der oberen Zeile zeigen einen größeren Sampleradius als die Abbildungen 3.10e bis 3.10h darunter.

4 Eigene Ideen zur Weiterentwicklung von SSAO-Verfahren

4.1 SSAO aus Sicht der Lichtquelle

Bei der Berechnung von direkten Schatten mit Hilfe von Shadow-Mapping wird die Tiefenkarte aus Sicht der Lichtquelle erfasst. Im Zuge der Bachelor-Arbeit wurde geprüft, inwiefern diese Informationen für Algorithmen die im Screen-Space (aus Sicht der Kamera) arbeiten nutzbar sind oder ob sich das Erstellen einer Tiefenkarte aus Sicht der Kamera unter Umständen einsparen lässt.

4.1.1 Implementierung

Da die Tiefenkarte nur aus Sicht einer Lichtquelle gegeben ist, müssen die Koordinaten aus Sicht der Kamera in das Textur-Koordinatensystem der Lichtquelle umgerechnet werden. Diese Transformation erfolgt wie beim Shadowmapping durch Multiplikation der Vertices im Kamerakoordinatensystem mit der Matrix $M = bias * lightProj * lightView * camView^{-1}$. [NFHS11, Kapitel 14.1.1 Schatten-Texturierung]

Es wurde folgender Algorithmus implementiert:

- Rendern der Tiefenkarte aus Sicht der Lichtquelle
- Rendern der Szene aus Sicht der Kamera
- Rendern von SSAO mit der Tiefenkarte aus Sicht der Lichtquelle

4.2 Dynamische Skalierung des SSAO Sample-Radius

Loos und Sloan [LS10] zeigen, dass bei einem kleinen Sample-Radius feine Details in der Geometrie erfasst werden, dabei aber Kontaktschatten verloren gehen, da zu deren Berechnung ein größerer Sample-Radius nötig ist. Durch das doppelte Abtasten des Image-Space mit einem kleinen und einen großen Radius können sowohl feine Details als auch Kontaktschatten ermittelt werden.

Dass beim Berechnen von SSAO Details verloren gehen liegt an der Echtzeitanforderung, durch die nur eine begrenzte Anzahl an Samples erhoben werden kann. Würden beliebig viele Samples verwendet, könnten sowohl feine Details als auch weitläufige Kontaktschatten zusammen berechnet werden.

Unsere Idee besteht darin, den Sample-Radius dynamisch zu skalieren um durch einmaliges Abtasten sowohl kleine Details als auch weitläufige Kontaktschatten zu berechnen. Zu diesem Zwecke soll die Umgebung um P vor der Occlusion-Berechnung erst sondiert und anhand eines geeigneten Kriteriums ein Skalierungsfaktor für den SSAO-Radius bestimmt werden.

Auf diese Weise kann der Radius für jeden SSAO-Algorithmus angepasst werden, da die Sondierung unabhängig vom danach verwendeten Verfahren ist.

Eine Gemeinsamkeit der SSAO-Verfahren ist, dass der Sampling-Radius willkürlich gewählt wird. Er wird als Parameter an den Fragmentshader übergeben und hängt von der dargestellten Geometrie ab. Ist eine Szene sehr groß, dann muss auch der Sample-Radius groß gewählt werden und umgekehrt. Durch die dynamische Skalierung könnte eine Anpassung des Sample-Radius anhand der Geometrie automatisch erfolgen.

4.2.1 Mathematisches Modell

Mit Hilfe eines mathematischen Modells des Line-Sampling Algorithmus von [LS10] konnten mehrere Geometrieeigenschaften simuliert werden, die in Tabelle 4.1 abgebildet sind. Die Spalte Überlagernde Frequenzen simuliert eine Geometrie, die wegen der groben Grundstruktur für die Kontaktschatten einen Radius von 3 Einheiten benötigen würde. Um die Details zu ermitteln wäre jedoch ein Radius von 0,25 nötig, bei dem aber der weitläufige Schatten der Grundstruktur verloren geht. Loos und Sloan [LS10] lösen dieses Dilemma beim Dual-Radii-Verfahren indem beide Radien ausgewertet und deren Ergebnisse verrechnet werden [2].

Der Line-Sampling Algorithmus wurde in Mathematica 8.0 im Zweidimensionalen umgesetzt. Die x- und y-Achsen wurden dabei zusammengelegt. Die Tiefenkarte konnte auf diese Weise als Funktion definiert und die Occlusion-Werte in Abhängigkeit dieser berechnet werden. Dabei wurde darauf geachtet Undersampling [3] und zufällige Rotation [4] zum Erzeugen mehrerer Sample-Konstellationen zu simulieren.

Am Beispiel des Ringing in Tabelle 4.1a kann man sehen, dass es falsch wäre, den Radius immer zu verkleinern. Bei den Funktionsmaxima ist es ohne Probleme möglich den Radius zu verringern, da keine Geometrie benachbart ist, die dadurch vernachlässigt würde. In den Funktionstälern ist das verkleinern des Radius verlustbehaftet. Die weitläufigeren Schatten gehen verloren.

Die letzte Spalte in Tabelle 4.1a in der Zeile mit dem Radius 0,25 zeigt, dass der Radius nicht zu klein gewählt werden darf, da sonst in den Tälern der Funktion der Occlusion-Wert fälschlicher Weise heller wird. Wegen des zu geringen Radius werden die Täler als flach wahrgenommen. Anhand des Radius 3 zeigt sich, dass der Radius auch nicht zu groß gewählt werden darf, da sonst die Schatten verblassen. Es lässt sich also schlussfolgern, dass der Radius von der Geometrie abhängig ist und nicht allgemein gültig festgelegt

[2] Entweder werden beide Werte multipliziert, oder das Minimum verwendet

[3] Es werden nur 3 Samples ausgewertet

[4] Rotation meint hier das horizontale verschieben der Line-Samples innerhalb des Kreises um $\pm 0,2 \cdot$ *Sample-Radius*

werden kann.

Der Vergleich von 4.1b mit 4.1a zeigt, dass durch zwei Durchgänge mit unterschiedlichen Radien mehr Details aus der Geometrie gewonnen werden können. 4.1c zeigt, dass auch bei einem Durchgang mit entsprechender Skalierung des Radius mehr Details gewonnen werden können.

Anhand des mathematischen Modells konnte gezeigt werden, dass durch die Skalierung des Sample-Radius an den richtigen Stellen mehr Details aus der Tiefenkarte extrahiert werden können. Umgekehrt können auch weitläufige Schatten verloren gehen, wenn an falschen Stellen skaliert wird.

4.2.2 Algorithmen

Als erster Ansatz sollte die gesamten zur Verfügung stehenden Informationen -also jedes Fragment- um den Punkt P betrachtet werden. Die Suche in der Umgebung mittels full search hat sich als nicht durchführbar erwiesen, da durch die vielen Samples das Programm bei der Suche nach z_{max} einfror. Daher wurde eine Poisson-Disk mit hinreichend Samples (328) für die Suche verwendet.

Die folgenden Algorithmen wurden auf ihre Qualität überprüft:

Basis-Algorithmus

- Rendere Tiefenkarte aus Sicht des Betrachters
- Berechne r_{neu} mit einem der folgenden Algorithmen
- Berechne SSAO mit r_{neu}

Algorithmus 1: Maximaler Z-Wert

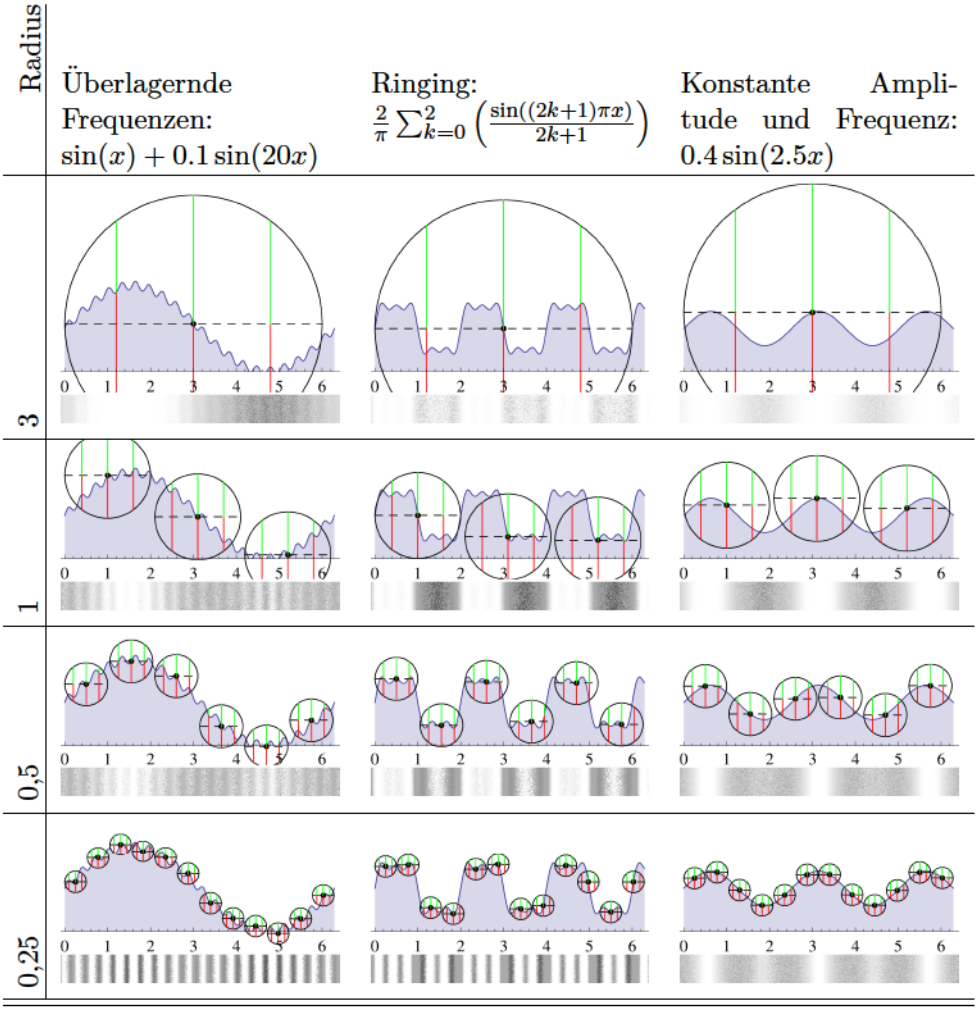
- Suche z_{max} in Umgebung um P mit Radius $r_0\left(z\left(P\right)\right)$
- Berechne $r_{neu} = \frac{z - z_{max}}{z_{max}} \cdot r_0\left(z\left(P\right)\right)$

Algorithmus 2: Mittlerer Z-Wert

- Suche die durchschnittliche Tiefe z_{avg} in Umgebung um P mit Radius $r_0\left(z\left(P\right)\right)$ für alle $z_{sample} > z(P)$
- Berechne $r_{neu} = \frac{z - z_{avg}}{z_{avg}} \cdot r_0\left(z\left(P\right)\right)$

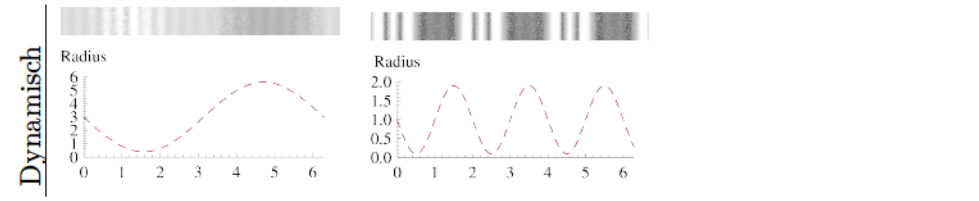
Algorithmus 3: Standardabweichung

- Suche die Standardabweichung $z_{std} = \sqrt{\sum_{i=0}^{n=328} \frac{\left(z(n) - z(P)\right)^2}{n}}$ in Umgebung um P mit Radius $r_0\left(z\left(P\right)\right)$
- Berechne $r_{neu} = \frac{z - z_{std}}{z_{std}} \cdot r_0\left(z\left(P\right)\right)$



(a) Fester Radius

(b) Zum Vergleich mit dem dynamischen Radius wurden die Werte der Radien 0,25 und 1 multipliziert.



(c) Die Graphen zeigen den verwendeten dynamischen Radius. Darüber ist das Ergebnis der Berechnung mit dynamischem Radius abgebildet.

Tabelle 4.1: Line-Sampling über eine 2D-Tiefenkarte. Unterhalb der Funktion die die Geometrie simuliert ist der Occlusion-Wert eingezeichnet. Dieser wurde ermittelt, indem der Line-Sample Kreis mit bestimmten Radius auf der Funktion platziert und die grün eingezeichneten Linienanteile ausgewertet wurden. Die horizontale Achse der Occlusion-Werte steht in Relation zur dargestellten Geometrie-Funktion. In der vertikalen sind mehrere verschiedene Durchläufe der selben Occlusion-Berechnung zu sehen, wobei die x-Position der Line-Samples zufällig innerhalb des Kreises variiert.

Algorithmus 4: Maximaler z-Wert mit maximaler z-Differenz

- Suche z_{max} in Umgebung um P mit Radius $r_0\left(z\left(P\right)\right)$
- Wenn $abs\left(z(P) - z_{max}\right) < z_{maxdiff}$ dann
  - Berechne $r_{neu} = \frac{z - z_{max}}{z_{max}} \cdot r_0\left(z\left(P\right)\right)$
- Sonst $r_{neu} = r_0(z(P))$ (Radius bleibt gleich)

Algorithmus 5: Maximaler z-Wert mit Vektorlänge und maximaler z-Differenz

- Initialisiere $\vec{z}_{max} = \begin{pmatrix} 0 & 0 & -far \end{pmatrix}^T$
- Für alle z_{sample} in Umgebung $\begin{pmatrix} u & v \end{pmatrix}^T$ um $\vec{P}$ mit Radius $r_0\left(z\left(\vec{P}\right)\right)$
  - Wenn $z_{sample} <= \left|z_{max} \cdot \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T\right|$ und $abs\left(z_{sample} - z\left(\vec{P}\right)\right) < z_{maxdiff}$
  - Dann setzte $\vec{z}_{max} = \vec{sample}$
- Berechne $r_{neu} = \left|\vec{z}_{max} - \vec{P}\right|$

Algorithmus 6: Maximaler z-Wert mit Vektorlängenprüfung

- Wie Algorithmus 5 aber ohne maximaler z-Differenz und mit zusätzlicher Bedingung. Die Samples müssen innerhalb des View-Space Sample-Radius R liegen.
  - Wenn $z_{sample} > \left|\vec{z}_{max} \cdot \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T\right|$ und $\left|\vec{sample} - \vec{P}\right| < R$ und $\left|\vec{sample}\right| < |\vec{z}_{max}|$
  - Dann setzte $\vec{z}_{max} = \vec{sample}$

5 Implementierung aktueller SSAO-Verfahren

Die Implementierung der SSAO-Algorithmen erfolgt mit OpenGL als C++ Programm. Das Programm führt mehrere Renderschritte durch:

1. Rendern der Szene um die lineare Tiefenkarte und Normalen zu erhalten
2. Rendern des ausgewählten SSAO Algorithmus
3. Rendern der Szene inklusive Beleuchtung (Phong'sches Beleuchtugnsmodell) und anwenden des berechneten Occlusion-Werts oder direktes Anzeigen des Occlusion-Werts

5.1 Screen-Space Ambient Occlusion

Der SSAO Algorithmus wurde in den beiden Varianten SSAO-Scheibe und SSAO-Kugel implementiert, die im folgenden beschrieben werden.

5.1.1 SSAO-Scheibe

Der SSAO-Scheibe Algorithmus wurde größtenteils von Kajalin übernommen [Kaj09]. Abweichend davon wird an Stelle der Rotation der Samples eine Spiegelung dieser vorgenommen. Zu diesem Zweck wird eine Noise-Textur der Größe 4x4 an den Fragmentshader übergeben. Die Noise-Textur wird über den gesamten Screen-Space wiederholt. Das im Original rotierte Sample "vRotatedOffset"kann nun durch Spiegelung mit folgendem GLSL-Code erzeugt werden:

```
vec3 noise = 2.0 *  texture(noise_tex,  noise_uv ).xyz - 1.0;
vec3 vRotatedOffset = reflect(vOffset, normalize(noise));
```

Die Rotationsmatrix aus [Kaj09] entfällt.

5.1.2 SSAO-Kugel

Der SSAO-Kugel Algorithmus wurde von Firbach [Fir09] übernommen und konnte größtenteils beibehalten werden.

5.1.3 Abweichungen vom Algorithmus

Kajalin [Kaj09] geht von einer Default-Occlusion von 50% bei ebenen Flächen aus. Bei Ecken ist weniger als 50% der Samples verdeckt, was dazu führt, dass Ecken heller erscheinen als Flächen. Dieses „Edge highlighting "wurde von Crytek bewusst belassen um dem Betrachter mehr Geometrieeigenschaften erkannbar zu machen.

In unserer Implementierung wurden alle Occlusion-Werte größer als 50% verworfen und die Werte zwischen 0% und 50% mit dem Faktor 2 skaliert. Dadurch werden ebene Flächen als Weiß dargestellt und es ist kein „Edge highlighting “mehr zu sehen.

5.2 Volumetric Obscurance

Bei der Implementierung von VO wurde der Beispielcode aus der Präsentation [JPO10] von ToyStory 3 verwendet.

Die Samples wurden mit Hilfe von Mathematica 8.0 erzeugt. Dabei wurde darauf verzichtet die Samples so anzuordnen, dass diese dieselben Volumen haben. Die Samples wurden von Hand auf der Einheitsscheibe verteilt und für alle Samples das Volumen berechnet. Am Ende wurde GLSL-Code erzeugt, der die Sample-Position auf der Scheibe, die zugehörige Line-Sample-Länge z_s und das berechnete Volumen des Samples enthält. Im Idealfall hätten alle Samples das gleiche Volumen und es müsste nicht im GLSL-Code abgelegt werden. In unserem Fall wurden die berechneten Line-Samples mit dem assoziierten Volumen gewichtet. Die Performanz und visuelle Qualität könnte an dieser Stelle noch weiter ausgebaut werden.

5.3 Image-Space Horizon-Based Ambient Occlusion

Bei der Implementierung wurde der Beispielcode [LB13b] aus dem DirectX10 SDK von Nvidia als Ausgangsmaterial verwendet. Da es sich um eine DirectX Anwendung handelt, musste der Code zu OpenGL portiert werden.

Der Shader-Code konnte dabei größtenteils übernommen werden. Es musste jedoch an einigen Stellen, die Tabelle 5.1 zu entnehmen sind, angepasst werden.

Außerdem musste beachtet werden, dass die Koordinatensysteme [5] bei OpenGL anders als bei Direct3D sind. Aus der HLSL Variante:

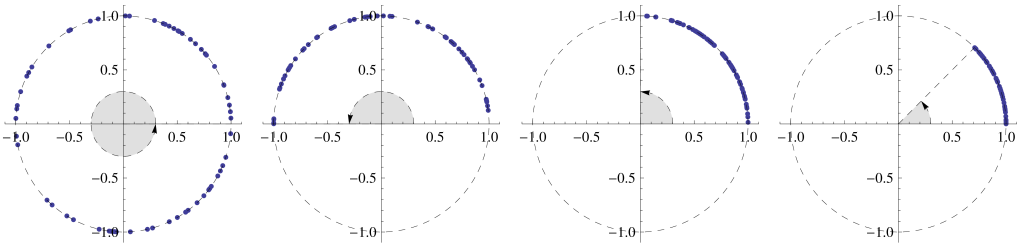
```
float3 uv_to_eye(float2 uv, float eye_z)
{
    uv = (uv * float2(2.0, -2.0) - float2(1.0, -1.0));
    return float3(uv * g_InvFocalLen * eye_z, eye_z);
}
```

wurde in GLSL:

```
vec3 uv_to_eye(vec2 uv, float eye_z) {
    return vec3(g_InvFocalLen * (uv * 2.0 - 1.0) * -eye_z, -eye_z);
}
```

Auf diese wurde im restlichen GLSL-Code mit positiven Z werten gerechnet.

[5] In diesem Fall Texturespace und Viewspace



(a) Eine Richtung (360°) (b) Zwei Richtungen (180°) (c) Vier Richtungen (90°) (d) Acht Richtungen (45°)

Abbildung 5.1: Rotationsverteilungen mit jeweils 64 Beispielwerten, die auf dem Einheitskreis verteilt dargestellt werden. Wird HBAO nur in eine Richtung abgetastet, dann wird in der Rotationstextur der volle Einheitskreis abgedeckt. Bei zwei Richtungen nur noch die Hälfte des Kreises, bei vier nur noch ein Viertel usw.

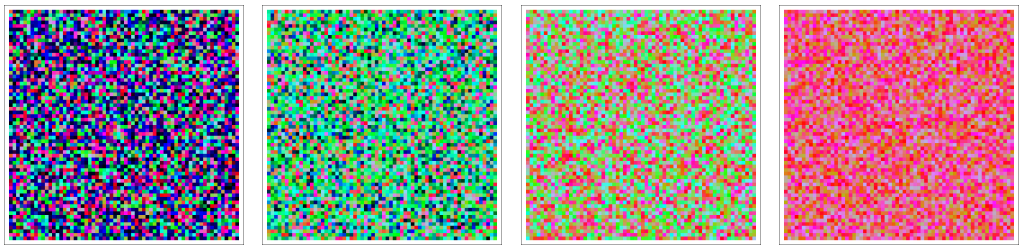
Beim C++-Code des Programms konnte nur die Header-Datei 'MersenneTwister.h', die zum Erzeugen von Zufallswerten verwendet wird, direkt durch Kopieren übernommen werden. Ansonsten wurde jeweils der Algorithmus des Quellprogramms analysiert und entsprechend im eigenen Zielprogramm umgesetzt.

- Erzeugen einer Textur der Größe 64x64 mit dem Pixelformat GL_RGB16F, die Zufallswerte für Rotation und Jitter enthält. Dabei enthält der Rot-Kanal $cos(\alpha)$, der Grün-Kanal $sin(\alpha)$, α ist ein Zufallswert und der Blau-Kanal einen anderen Zufallswert für den Jitter. Die Rotationstextur ist abhängig von der Anzahl der verschiedenen Richtungen des HBAO-Algorithmus. Deshalb wird sie jedes mal neu erzeugt, wenn sich die Anzahl der Richtungen ändert.

  Wie in Abbildung 5.1 zu sehen ist, wird in der Rotationstextur nur so viel des Einheitskreises abgedeckt wie unbedingt nötig ist. Tastet der Algorithmus nur in einer Richtung die z-Werte ab, so müssen die vollen 360° in der Textur abgelegt werden. Wenn der Algorithmus in mehr Richtungen abtastet, macht es wenig Sinn, zusätzliche Samples in eine Richtung zu erzeugen, die sowieso schon durch den Algorithmus abgetastet wird.

  Wird zum Beispiel in die Richtungen 0°, 90°, 180° und 270° abgetastet, dann reicht es aus, Werte zwischen 0° und 90° ($\frac{360°}{4}$) in der Rotationstextur zu hinterlegen.

- Setzen der Shader-Parameter wie z.B. Auflösung, Bildseitenverhältnis, Anzahl der Schritte, Anzahl der Richtungen etc.



(a) Eine Richtung (360°) (b) Zwei Richtungen (180°) (c) Vier Richtungen (90°) (d) Acht Richtungen (45°)

Abbildung 5.2: Rotationstextur bei HBAO der Größe 64x64 Pixel. Je mehr Richtungen der Algorithmus abdeckt, desto gleicher werden die Farben in der Textur, da die Rot und Grün Werte immer näher beieinander liegen.

5.4 Sample-Radius

In Tabelle 6.1 ist die beim Raycaster-Programm verwendete Strahllänge angegeben. Um die Performanz[6] und Qualität der implementierten SSAO-Algorithmen korrekt zu vergleichen, muss die Strahllänge beim Raytracing möglichst genau auf den Sample-Radius abgebildet werden.

Zu diesem Zweck wurden alle Algorithmen so angepasst, dass der Sample-Radius im Viewspace gegeben ist. Bei HBAO, SSAO-Kugel und VO war dies bereits der Fall, wobei die SSAO-Kugel und VO Implementierung aus Gründen der Einheitlichkeit ebenfalls angepasst wurden.

5.4.1 Herleitung der HBAO Sampleradius-Transformation

In der Implementierung [LB13b] erfolgt die Transformation des Sample-Radius vom Viewspace in den Texturespace durch folgenden GLSL-Code:

```
vec2 rad_texturespace = 0.5 * g_R  * g_FocalLen / P.z ;
```

Dabei entspricht der Parameter g_R dem Sample-Radius, $P.z$ ist der z-Wert des Sample-Punktes und $g\_FocalLen$ ist $\left(\frac{1}{\alpha}\cot\frac{fovy}{2} \quad \cot\frac{fovy}{2}\right)^T$, wobei α das Bildseitenformat und $fovy$ der Winkel des vertikalen Gesichtsfeldes ist.

Diese Formel kann wie folgt hergeleitet werden. Laut [ESAW11, B.1.3] ist die OpenGL-Projektionsmatrix

$$M_p = \begin{pmatrix} \frac{1}{\alpha}\cot\frac{fovy}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{5.1}$$

[6] Ein größerer Radius sorgt für eine geringere Performanz, da die Zugriffe auf die Tiefenkarte gepuffert werden. Bei einem zu großen Sample-Radius befinden sich die die gewünschten Werte nicht mehr im Cache.

Als Ausgangspunkt sei der Punkt $v = \begin{pmatrix} x & y & z & 1 \end{pmatrix}^T$ gegeben. Der projizierte Punkt $v_p = M_p \cdot v$ ist nach der Normierung

$$v_p = \begin{pmatrix} -\frac{x \cot\left(\frac{fovy}{2}\right)}{\alpha z} & -\frac{y \cot\left(\frac{fovy}{2}\right)}{z} & \frac{2fn+fz+nz}{fz-nz} \end{pmatrix}^T$$

und befindet sich im Clipspace, bei dem die Werte für x, y und z von -1 bis 1 reichen. Die Projektion des z-Wertes ist hier nicht linear gegeben, was für die Herleitung aber nicht von Bedeutung ist.

Die Transformation des Punktes v_p in den Texturspace, bei dem die Werte für x, y und z von 0 bis 1 reichen, erfolgt über

$$v_t = \frac{(v_p + 1)}{2} = \begin{pmatrix} \frac{1}{2} \cdot \left(1 - \frac{x \cot\left(\frac{fovy}{2}\right)}{\alpha z}\right) \\ \frac{1}{2} \cdot \left(1 - \frac{x \cot\left(\frac{fovy}{2}\right)}{z}\right) \\ \frac{f(n+z)}{(f-n)z} \end{pmatrix} \tag{5.2}$$

Liegt v_p auf der Near-Plane, dann ist der z-Wert des transformierten Punktes v_t gleich 0. Liegt er auf der Far-Plane ist der transformierte z-Wert gleich 1.

Nun ist bekannt wie ein beliebiger Punkt im Viewspace in den Texturspace transformiert werden kann. Wenn des Bildseitenformat nicht 1 zu 1 ist, wird ein Kreis bei der Transformation auf eine Ellipse abgebildet. Daher muss zwischen einem horizontalen und einem vertikalen Radius unterschieden werden.

Um den horizontalen Radius zu berechnen werden zwei Punkte v_{p1} und v_{p2} im Eyespace mit $\begin{pmatrix} d & 0 & z \end{pmatrix}^T$ und $\begin{pmatrix} -d & 0 & z \end{pmatrix}^T$ angenommen. Ist $z = -n$, dann befinden sich die Punkte auf der Near-Plane an der Position d-Einheiten links bzw. rechts vom Ursprung. Je näher sich der z Wert an die Far-Plane annähert, desto kleiner wird der Horizontale Abstand zwischen den Punkten. Dabei entspricht d dem horizontalen Durchmesser im Viewspace.

Nun werden die beiden Punkte in den Texturspace transformiert. Die Differenz $v_{p2} - v_{p1}$ ergibt den horizontalen Durchmesser d im Texturespace. Um den Radius zu berechnen muss nur noch $d = \frac{r}{2}$ substituiert werden.

$$v_{p2} - v_{p1} = \begin{pmatrix} d \cdot \frac{\cot\left(\frac{fovy}{2}\right)}{\alpha \cdot z} & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} \frac{r}{2} \cdot \frac{\cot\left(\frac{fovy}{2}\right)}{\alpha \cdot z} & 0 & 0 \end{pmatrix}^T$$

Vertikal wird mit den Punkten $v_{p3} = \begin{pmatrix} 0 & d & z \end{pmatrix}^T$ und $v_{p4} = \begin{pmatrix} 0 & -d & z \end{pmatrix}^T$ analog verfahren, um den vertikalen Texturspace-Radius zu erhalten.

$$v_{p4} - v_{p3} = \begin{pmatrix} 0 & \frac{r}{2} \cdot \frac{\cot\left(\frac{fovy}{2}\right)}{z} & 0 \end{pmatrix}^T$$

Vertikaler und horizontaler Radius zusammen ergibt daher den folgenden Vektor

$$rad\_texturespace = \begin{pmatrix} \frac{r}{2} \cdot \frac{\cot\left(\frac{fovy}{2}\right)}{\alpha \cdot z} \\ \frac{r}{2} \cdot \frac{\cot\left(\frac{fovy}{2}\right)}{z} \end{pmatrix} \tag{5.3}$$

der in [LB13b] durch folgende Zeile gegeben ist:

```
vec2 rad_texturespace = 0.5 * g_R  * g_FocalLen / P.z ;
```

$0.5 * g\_R$ entspricht der oben erwähnten Substitution $d = \frac{r}{2}$. $g\_FocalLen$ erhält man durch ausklammern des z-Wertes und kann mit der CPU vorausberechnet und an den Fragmentshader als Parameter übergeben werden.

5.4.2 Anpassen der Implementierungen

HBAO musste nicht angepasst werden, da diese Implementierung als Referenz diente.

Bei Volumetric Obscurance können die auf der Einheitsscheibe verteilten Samples durch Multiplikation mit der Formel 5.3 mit einem gegebenen Viewspace-Radius r in den Texturspace transformiert werden.

Die SSAO-Sphäre konnte einfach angepasst werden. Die Sampleoffset-Koordinaten waren bereits im Viewspace gegeben. Um den z-Wert der Sampleoffsets zu ermitteln wurde der Sampleoffset mit Hilfe der Projektionsmatrix in den Clipspace transformiert und dann in Texturkoordinaten umgewandelt:

```
vec2 project_to_screenspace(vec3 viewSpacePos) {
    vec4 projected = proj * vec4(viewSpacePos, 1.0);
    return (projected.xy / projected.w + vec2(1.0)) * 0.5;
}
```

An Stelle der Multiplikation mit der gesamten Projektionsmatrix aus Formel 5.1 können beim Ermitteln der Texturkoordinaten auch lediglich die x- und y-Komponenten des Vektors aus Formel 5.2 betrachtet werden. Es ergibt sich daher folgende Implementierung für die Transfomation vom Viewspace zum Texturespace:

```
vec2 eyespace_to_texturespace(vec3 v) {
    return vec2(
        0.5f * (1 - ((v.x / v.z) * g_FocalLen.x)),
        0.5f - g_FocalLen.y * v.y / (2.0f * v.z)
    );
}
```

SSAO-Scheibe beinhaltet, so wie es in [Kaj09] definiert wurde, keinerlei Skalierung des Radius. Das ist der Hauptunterschied dieses Algorithmus zur SSAO-Sphäre.

5.5 Samplegenerierung

Zum Vergleich wurden Bilder mit 8, 16 und 24 Samples mit allen implementierten Algorithmen erstellt. Da die Vorlage für den Algorithmus SSAO-Kugel von Firbach [Fir09] nur einen 128 Samples großen Kernel enthielt, mussten entsprechend neue Samples generiert werden. Wie bei [Mit07] wurde auch in der Implementierung von Herrn Firbach die quadratische Abschwächung des Lichts nach Distanz durch die Verteilung der Samples approximiert. Salopp gesagt befinden sich in der Mitte der Sphäre mehr Samples als außen.

Die Funktion $f(x)$ (Gleichung 5.4) hat als Parameter x die Distanz eines Punktes zum Mittelpunkt der Sphäre. Als Rückgabewert liefert sie einen Wert, der bei $f(0)$ gleich 1 ist und quadratisch abnimmt, bis bei $f(1)$ der Wert 0 angenommen wird. Um nun aus der Funktion eine Wahrscheinlichkeitsverteilungsfunktion $d(x)$ herzuleiten, muss $f(x)$ so skaliert werden, dass die Summe aller Wahrscheinlichkeiten 1 ergibt (Gleichung 5.5). Durch Multiplikation der Funktion $f(x)$ mit dem Faktor a erhält man die Funktion $d(x)$.

$$f(x) = \begin{cases} 0 & für \quad x < 0 \\ 1 - x^2 & für \quad x \geq 0 \\ 0 & für \quad x \geq 1 \end{cases} \tag{5.4}$$

$$1 = a \cdot \int_{-\infty}^{\infty} f\,(x) \tag{5.5}$$

$$a = \frac{3}{2} \tag{5.6}$$

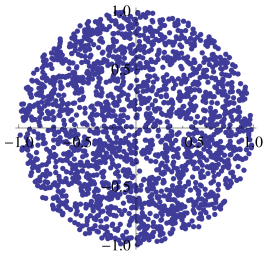
$$d(x) = a \cdot f(x) \tag{5.7}$$

Das Tutorial [DG13a] wurde als Ausgangsmaterial verwendet. Es zeigt wie mit Mathematica zufällig erzeugte Punkte auf einer Scheibe gleich verteilt werden. Die Punkte werden wie in Algorithmus 1 erzeugt.

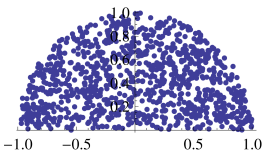
Algorithmus 1 : Erzeugen zufälliger Punkte innerhalb einer Scheibe, so dass die Punkte in dieser gleichverteilt sind.

1 $\phi \leftarrow$ zufälliger Winkel gleichverteilt $\in [0, 2\pi)$;
2 $u \leftarrow$ zufälliger Wert gleichverteilt $\in [0, 1]$;
3 $P \leftarrow \begin{pmatrix} \sqrt{u}\sin(\phi) \\ \sqrt{u}\cos(\phi) \end{pmatrix}$;
4 **return** P ;

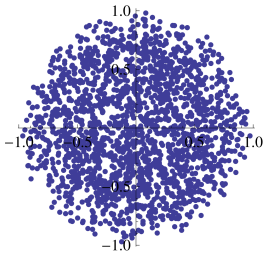
Abbildung 5.3a zeigt das Ergebnis für 2000 zufällig erzeugte Punkte. Wird der Wertebereich des Winkels ϕ auf $[0, \pi]$ verringert, so erhält man als Ergebnis gleichverteilte Punkte für eine halbe Scheibe wie es in Abbildung 5.3b zu sehen ist. Im selben Algorithmus kann für den Wert u eine eigene Wahrscheinlichkeitsverteilung verwendet werden. Durch Verwenden der Funktion $d(x)$ werden mehr Samples im Zentrum der Scheibe erzeugt. Das Ergebnis ist in 5.3c zu sehen.



(a) 2000 gleichverteilte Punkte innerhalb des Einheitskreises.



(b) 1000 gleichverteilte Punkte innerhalb des halben Einheitskreises.



(c) 2000 nach Funktion $d(x)$ verteilte Punkte innerhalb des Einheitskreises.

Abbildung 5.3: Verteilung der Samples innerhalb der Scheibe mit verschiedenen Wahrscheinlichkeiten

Im nächsten Schritt wurde eine weitere Dimension zu dem Algorithmus hinzugefügt um Punkte in einer Sphäre zu erhalten. Als erster Ansatz wurden die Winkel θ und ϕ aus 2π und π gewählt. Dies führt jedoch zu einer Anhäufung von Punkten an den Stellen, an denen die x und y Koordinaten gegen 0 gehen, wie in den Abbildungen 5.4a und 5.4b zu sehen ist.

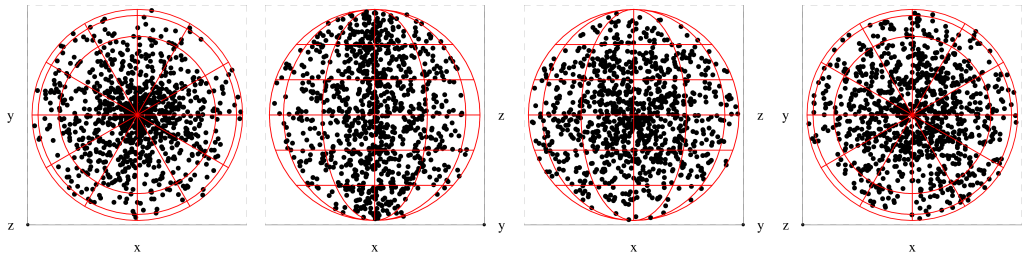
Die Korrekte Verteilung kann nach [DG13b] mit Algorithmus 2 erreicht werden.

Algorithmus 2 : Erzeugen zufälliger Punkte innerhalb einer Sphäre, so dass die Punkte in dieser gleichverteilt sind.

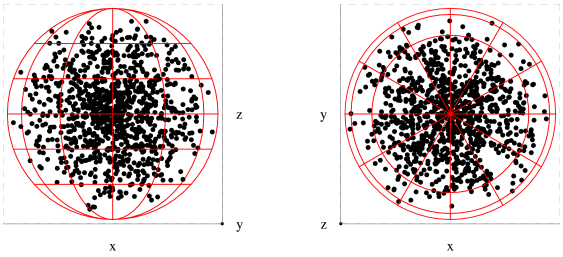
1 $u \leftarrow$ zufälliger Wert gleichverteilt $\in (0, 1)$;
2 $v \leftarrow$ zufälliger Wert gleichverteilt $\in (0, 1)$;
3 $w \leftarrow$ zufälliger Wert gleichverteilt $\in [0, 1]$;
4 $\theta \leftarrow 2\pi u$;
5 $\phi \leftarrow \cos^{-1}(2v - 1)$;
6 $P \leftarrow \sqrt{w} \begin{pmatrix} \cos(\theta)\sin(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\phi) \end{pmatrix}$;
7 **return** P ;

Auch in diesem Fall können die zufälligen Zahlen für w durch die Wahrscheinlichkeitsverteilungsfunktion $d(x)$ erzeugt werden. Ein beispielhaftes

Ergebnis ist den Abbildungen 5.4e und 5.4e zu sehen.



(a) Falsche Verteilung (b) Falsche Verteilung (c) Einheitliche Verteilung (d) Einheitliche Verteilung



(e) Quadratische Verteilung (f) Quadratische Verteilung

Abbildung 5.4: Verteilung der Samples innerhalb der Sphäre mit verschiedenen Wahrscheinlichkeiten. Bei der falschen Verteilung 5.4a und 5.4b ist eine Häufung von Punkten auf der z-Achse zu erkennen. Die Quadratische Verteilung 5.4e und 5.4f ergibt eine höhere Dichte in der Mitte der Sphäre, während bei der einheitlichen Veteilung 5.4c und 5.4d alle Punkte gleich über das gesamte Volumen verteilt sind.

Als Ergebnis konnten mit dem Algorithmus 2 mit der Wahrscheinlichkeitsfunktion $d(x)$ die Samples der SSAO-Sphäre und mit Algorithmus 1 mit Normalverteilung die Samples für die Scheibe der VO-Implementierung erzeugt werden.

Dieses Vorgehen ist für den VO Algorithmus nicht optimal, da [LS10] Zeit in eine optimale Sampleverteilung investiert haben, bei der das zugeordnete Volumen über alle Samples gleich verteilt ist. In dieser Arbeit wurde auf eine optimale Verteilung aus zeitlichen Gründen verzichtet. Sowohl die visuelle Qualität, als auch die Performanz [7] der VO-Implementierung könnte daher noch weiter verbessert werden.

[7] Da die Samples ungleiche Volumen haben, muss jedes Linesample einzeln nach dem zugeordneten Volumen gewichtet werden. Pro Sample muss daher eine zusätzliche Multiplikation ausgeführt werden. Bei gleichen Volumen würde dies entfallen.

Tabelle 5.1: Vergleich der Shadersprache HLSL (High Level Shader Language), die in DirectX verwendet wird, mit GLSL (OpenGL Shading Language)

Was	**HLSL**	**GLSL**
Datentypen	float2, float3, float4	vec2, vec3, vec4

Texturzugriff	Texture2D<float3> ran;
	Liest die Textur ohne Filterung
	`ran.Load(`
	`  int3((int)IN.pos.x,`
	`  (int)IN.pos.y, 0`
	`  )`
	`);`
		uniform sampler2D ran;
		Die Textur muss mit GL_NEAREST gesampelt werden.
		`texture(ran, uv);`

Parameter	`cbuffer cb0`
	`{`
	`  float g_Steps;`
	`  float g_Dir;`
	`  float g_R;`
	`  ...`
	`}`
		`uniform float g_Steps;`
		`uniform float g_Dir;`
		`uniform float g_R;`
		`...`

Funktionsnamen	ddx(In.EyePos.xyz) ddy(In.EyePos.xyz) rsqrt(dot(v,v))	dFdx(vsPosition) dFdy(vsPosition) inversesqrt(dot(v,v))

Normalen	`float3 N =`
	`  normalize(tNormal...);`
		`vec3 N = normalize(`
		`  texture( n_tex, uv )`
		`);`
		`N.z = -N.z;`

6 Ergebnisse

Im folgenden Kapitel werden die implementierten AO-Algorithmen visuell und anhand der Performanz verglichen.

6.1 Testsysteme und Vorraussetzungen

Die Performanz wurde mit folgenden Testsystemen ermittelt:

1.
   - Prozessor: 2.4GHz, E5620 Xeon, Intel
   - Arbeitsspeicher: 8GiB DDR2
   - Grafikkarte: GeForce GTX 680, Nvidia
   - Treiberversion: 314.07
   - Betriebssystem: Microsoft Windows 7 (SP1) 64 Bit
2.
   - Prozessor: 3.0 GHz, X9650 Core 2 Extreme, Intel
   - Arbeitsspeicher: 4GiB DDR2
   - Grafikkarte: Radeon HD 6970, AMD
   - Treiberversion: 13.1
   - Betriebssystem: Microsoft Windows 7 (SP1) 64 Bit

Alle Angaben zur Performanz sind in Millisekunden angegeben und wurden im Fenstermodus bei einer Auflösung von 1920x1080 Pixeln als Durchschnittlicher Wert über 100 erzeugte Einzellbilder ermittelt. Es wurde jeweils nur die Dauer der Ausführung des angegebenen Algorithmus ermittelt, das heißt die Erzeugung der Tiefenwerte und die Beleuchtung der Szene sind nicht enthalten. Die Messung der Zeit wurde mit GL_TIMESTAMP Queries durchgeführt.

Für den visuellen Vergleich wurden verschiedene Szenen mit teils unterschiedlichen Kameraperspektiven ausgewählt die in Tabelle 6.1 aufgelistet werden. Außerdem ist je Perspektive ein Referenzbild dargestellt, das mit Hilfe eines Raytracers erstellt wurde. Anhand der Referenzbilder können die visuellen Ergebnisse der Algorithmen bewertet werden.

6.2 Vergleich der implementierten SSAO Algorithmen

Zum Vergleich der Performanz wurden alle Fallunterscheidungen wie z.B.

```
if(use_noise)
  sample_offset = reflect(sample_offset, normalize(rand));
```

Die angegebene Anzahl an Samples entspricht der Anzahl an zusätzlichen Texturzugriffen auf die Tiefenkarte. Der erste Texturzugriff, mit dem die SSAO-Kugel im Raum bei Punkt P platziert wird, wurde bei keinem der

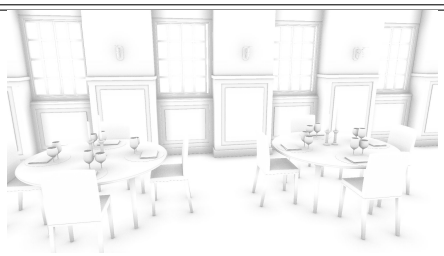
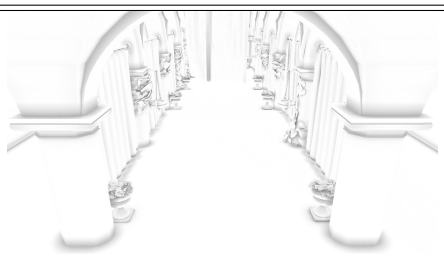
Szene	Raytracing Referenzbild	Strahllänge
Stanford Drache		4,8
Restaurant		0,12
Sponza 1		20,0
Sponza 2		20,0
Sponza 3		20,0

Tabelle 6.1: Szenenauswahl mit Referenzbildern, die mittels Raytracing mit 256 Strahlen und der angegebenen Strahlenlänge erstellt wurden.

Sample Anzahl	HBAO Richtungen	HBAO Schritte
8	4	2
16	4	4
24	6	4

Tabelle 6.2: Anzahl der Samples aus Tabelle 6.1 wird bei HBAO auf die Parameter Richtungen und Schritte abgebildet.

Algorithmen zur Sampleanzahl gezählt. Bei VO wird daher das Line-Sample in der Mitte der Einheitsscheibe nicht gezählt, da es per Definition immer zu 50% verdeckt ist und keinen zusätzlichen Texturzugriff erfordert. Bei HBAO ergibt sich die Sample-Anzahl wie in Tabelle 6.2 gelistet.

Die Screenshots von VO wurde mit Paired-Sampling erstellt, ohne Dual-Radii zu verwenden. Durch die Verwendung von Dual-Radii wären die Ergebnisse nicht vergleichbar, da hier die angegebene Anzahl an Samples pro Radius einmal ausgewertet wird. Es würden also doppelt so viele Samples wie bei den anderen Verfahren verwendet. Als maximale Occlusion-Distanz wurde der Sample-Radius * 2 verwendet. Dieser Wert wurde empirisch ermittelt. Bei HBAO wurde ein Angle-Bias von 30° verwendet und keine Normalen zur Berechnung verwendet[8].

Allgemein lässt sich an den Balkendiagrammen in Tabelle 6.1 sehen, dass VO bei 8 Samplen auf allen Testsystemen am schnellsten ist. Bei 16 bzw. 24 Samplen scheint VO und HBAO auf der HD 6970 gleich schnell. Bei der GTX680 mit 16 bzw. 24 Samples scheint HBAO schneller.

Die Geschwindigkeit von SSAO-Scheibe bzw. SSAO-Kugel ist weitgehend ähnlich. Bei 24 Samplen auf der GTX 680 ist SSAO-Scheibe teilweise [9] deutlich langsamer als SSAO-Kugel. Dies könnte daran liegen, dass der Radius der SSAO-Scheibe nicht abhängig vom z-Wert ist. Bei der Sponza Szene wird der Sampleradius bei der Kugel immer kleiner, je näher der Sampel-Mittelpunkt der Far-Plane kommt. Bei den anderen Fällen ist der Radius von SSAO-Scheibe und SSAO-Kugel vergleichbar und die Performanz ebenso.

Im Vergleich mit der Metrik „Root Mean Square Error“, im folgenden RMSE genannt, schneidet SSAO-Kugel wesentlich schlechter ab als SSAO-Scheibe. Eigentlich wurde erwartet, dass die Kugel das bessere Ergebnis liefert, da die Samples im View-Space verteilt wurden und die korrekte Transformation in den Texture-Space erfolgt. Der SSAO-Scheiben Algorithmus verzichtet darauf und verteilt die Samples eher willkürlich auf einer Scheibe

[8] Die Implementierung von Normalen wurde nur bei HBAO vorgenommen, da es bereits in der Vorlage [LB13b] enthalten war. Die anderen Algorithmen könnten durch das Verwenden von Normalen ebenso verbessert werden, worauf in dieser Arbeit jedoch verzichtet wurde.

[9] Sponza 1, Stanford Drache

(a) Bildausschnitt aus Nahaufnahme

(b) Bildausschnitt aus entfernter Aufnahme

Abbildung 6.1: Dieselbe Geometrie als Nahaufnahme und aus weiterer Entfernung. Weil die Geometrie bei 6.1b näher an der Far-Plane ist, wirft sie einen Schatten auf die dahinter liegende Wand. Nähert sich die Kamera dem Objekt an wird kein Schatten mehr geworfen. Dies ist in 6.1a zu sehen.

im Texture-Space. Durch die geringe Anzahl der Samples sind bei ebenen Flächen nach dem zufälligen Spiegeln häufig mehr Samples hinter der Fläche. Dadurch erscheint die Fläche insgesamt grau, obwohl sie eigentlich weiß sein müsste. Ab 24 Samplen werden auch bei der SSAO-Kugel ebene Flächen korrekt Weiß dargestellt.

Sponza 1 zeigt, dass sowohl bei SSAO-Kugel als auch Scheibe an den Säulen links und rechts ein falscher Schatten entsteht. Dies passiert wegen der Sample-Invalidierung, die abhängig von der Entfernung zur Kamera ist. Dies wurde in Kapitel 3.1.2 beschrieben. Der Schatten, den die Flagge in der Mitte des Bildes auf den Hintergrund wirft ist ebenso falsch und entsteht aus dem selben Grund. Abbildung 6.1 zeigt, wie sich die Distanz zur Kamera auf den Schattenwurf auswirkt.

VO und HBAO sind qualitativ wesentlich besser als die beiden anderen vorgestellten SSAO-Methoden, da sie aus den Samples mehr Informationen beziehen. Es wird nicht nur festgestellt, ob ein Sample verdeckt oder frei ist. Bei VO hängt die maximale Occlusion-Distanz von einem Parameter ab, der im Viewspace gegeben ist. Die Abschwächung ist also nicht von der Distanz zum Betrachter abhängig, wie es bei SSAO der Fall ist. Bei HBAO ist diese Distanz durch den Sample-Radius ebenfalls im Viewspace definiert. VO und HBAO schneiden im RMSE-Vergleich relativ gleich ab, wobei HBAO meistens leicht bessere Werte liefert. Durch die Verwendung des Angle-Bias kann HBAO Schattierungen, die durch geringe Tesselierung entstehen, vermeiden. Ein zu geringer Winkel, der kleiner als der Angle-Bias ist, wird nicht gewertet. Dies ist bei VO nicht möglich, da hier der Winkel zwischen den Samples nicht beachtet wird. Man könnte einen Angle-Bias jedoch implementieren, indem man den Winkel zwischen den Samplepaaren betrachtet. Als Beispiel kann die Szene Sponza 1 in Tabelle 6.1 betrachtet werden. Hier sind im oberen Bereich der Säulen links und rechts bei VO leichte Schatten zu erkennen, die bei HBAO durch den Angle-Bias wegfallen. Augenschein-

lich sind die Occlusion-Werte von HBAO näher an der Referenzlösung als es die von VO sind.

Da VO und HBAO wesentlich bessere Qualität bei vergleichbarer Performanz zu den restlichen Algorithmen liefern, sind sie zu bevorzugen. VO scheint seine Stärken bei wenigen Samples mit guter Performanz zu haben. Es hat jedoch den Nachteil, dass die Sample-Erzeugung relativ aufwändig und unflexibel ist. Man muss sich für eine bestimmte Anzahl und Verteilung von Samples entschieden und diese offline erzeugen und im Shader-Code hinterlegen. HBAO gestaltet sich hier flexibler, da dem Shader die Anzahl der Richtungen und Schritte übergeben werden. HBAO lässt sich flexibel für verschieden leistungsstarke Systeme anpassen, während VO sich eher anbietet, wenn die Anwendung möglichst performant sein soll.

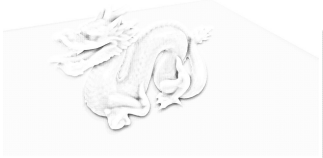
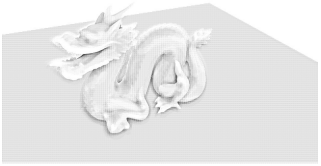
	Anzahl der Samples		
	24	16	8
SSAO-Scheibe			
	4.8% RMSE	4.9% RMSE	5.6% RMSE
SSAO-Kugel			
	4.8% RMSE	6.4% RMSE	23.7% RMSE
VO			
	2.3% RMSE	2.4% RMSE	2.5% RMSE
HBAO			
	2.4% RMSE	2.5% RMSE	2.8% RMSE

Tabelle 6.3 – Fortsetzung

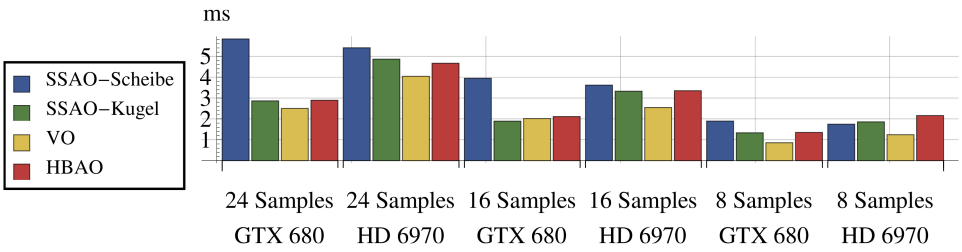
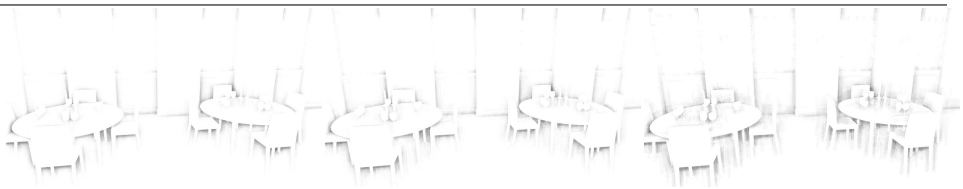
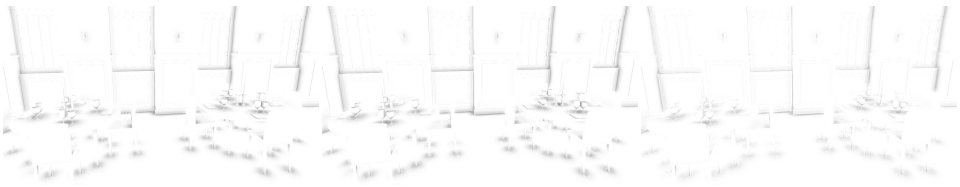
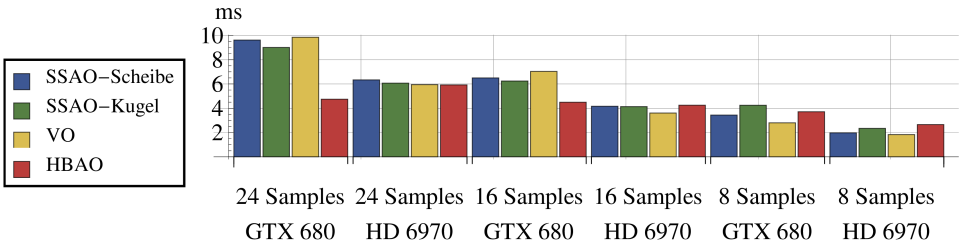
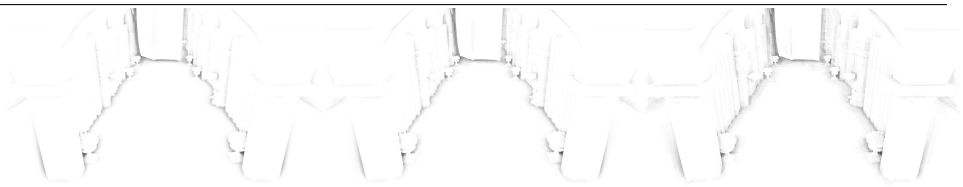
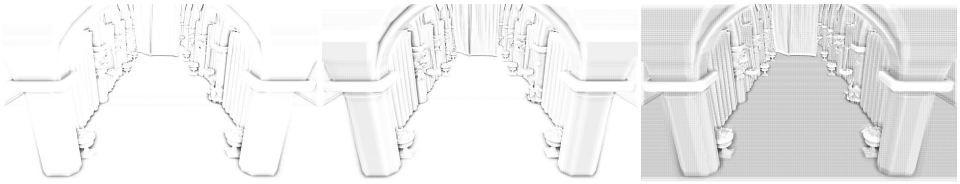
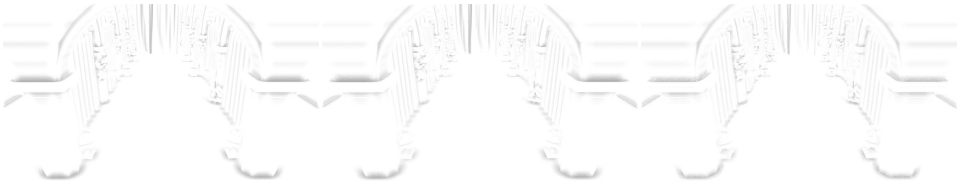
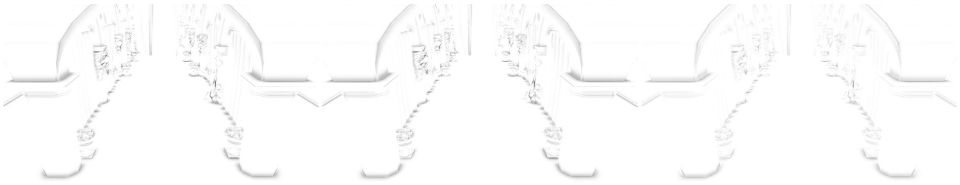
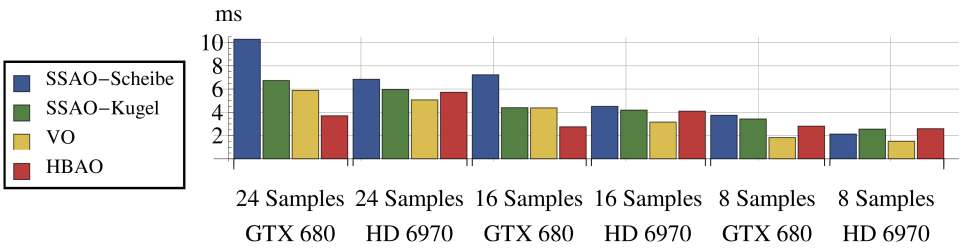
Anzahl der Samples		
24	16	8
		
SSAO-Scheibe		
	8.3% RMSE	8.6% RMSE	10% RMSE
SSAO-Kugel		
	9.5% RMSE	11.5% RMSE	26.4% RMSE
VO		
	6% RMSE	6.1% RMSE	6.2% RMSE
HBAO		
	5.9% RMSE	6.2% RMSE	6.4% RMSE
		

Tabelle 6.3 – Fortsetzung

	Anzahl der Samples		
	24	16	8
SSAO-Scheibe			
	7.7% RMSE	7.9% RMSE	8.7% RMSE
SSAO-Kugel			
	7.8% RMSE	10.4% RMSE	31% RMSE
VO			
	5.5% RMSE	5.6% RMSE	5.7% RMSE
HBAO			
	4.9% RMSE	5.1% RMSE	5.7% RMSE



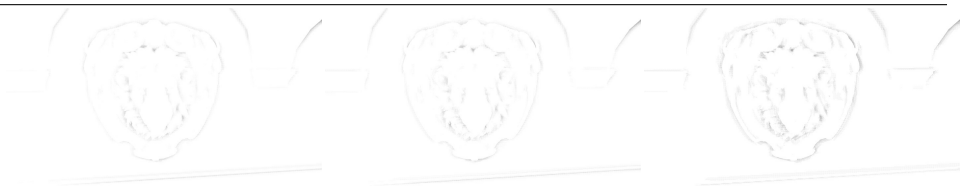
SSAO-Scheibe			
	7.2% RMSE	7.2% RMSE	7.7% RMSE

Tabelle 6.3 – Fortsetzung

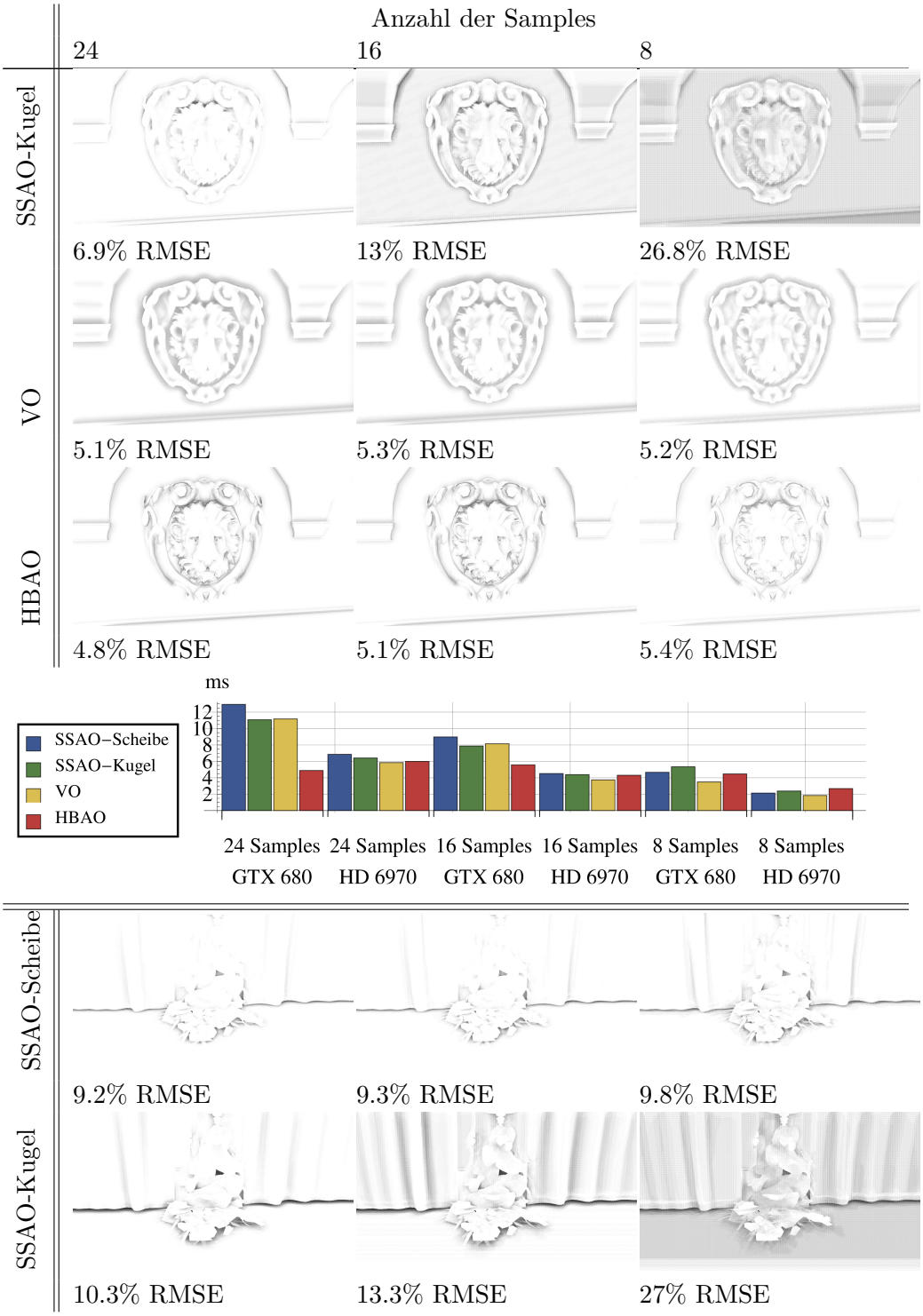
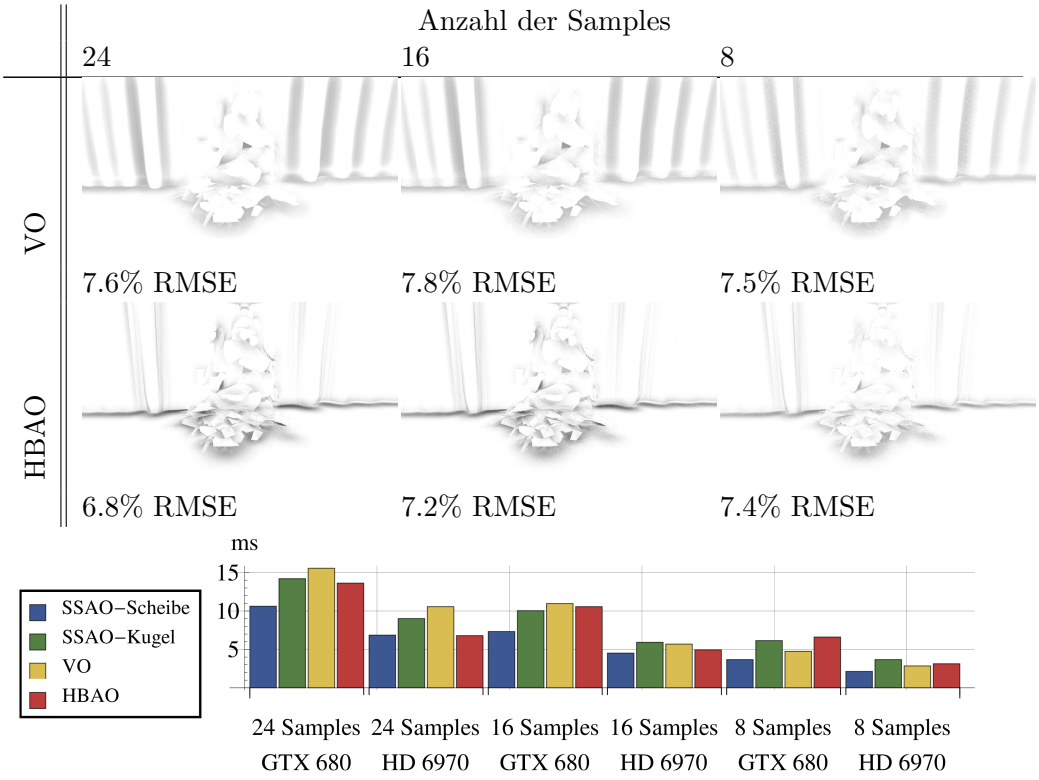


Tabelle 6.3 – Fortsetzung



6.3 SSAO aus Sicht der Lichtquelle

Die Tiefenkarte, die beim Shadow-Mapping erzeugt wird, kann nur in Spezialfällen die Tiefenkarte aus Sicht der Kamera ersetzen, da das View-Frustum aus der Sicht der Lichtquelle nicht immer das komplette View-Frustum der Kamera beinhaltet. Abbildung 6.2f zeigt, das die Tiefenkarte über dem Löwenkopf aufhört und durch Textur-Clamping gestreckt wird.

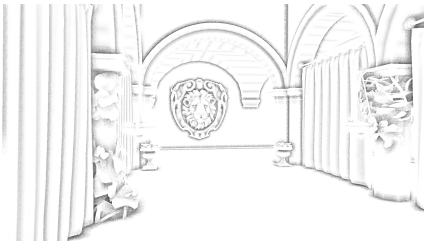
Selbst wenn das View-Frustum der Kamera komplett in dem des Lichts enthalten ist, muss die Kamera und die Lichtquelle in die selbe Richtung blicken. Ist auch diese Voraussetzung erfüllt, dann ist die Auflösung im Koordinatensystem des Lichts geringer, da die Near-Plane der Kamera nur zum Teil enthalten ist. Laut Bavoil und Sainz reicht für die Berechnung von SSAO die Hälfte der Auflösung der Tiefenkarte aus [BSD09].

Ist all das gegeben, dann kann man das in Abbildung 6.2d zu sehende Problem erkennen. Die Kamera-Koordinaten von der Wand hinter der Vase werden so in die Shadow-Mapping-Koordinaten transformiert, dass sie an die Stelle der Vase gelangen. Das hat zur Folge, dass die Vase doppelt abgebildet wird. Einmal an der korrekten Stelle wie sie in Abbildung 6.2c zu sehen ist, und einmal auf der Wand hinter der Vase wie in Abbildung 6.2d.

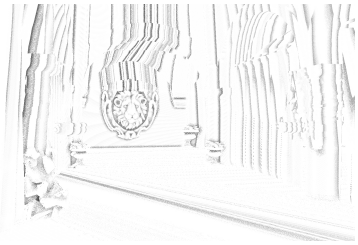
Es wären die folgenden Spezialfälle denkbar, in denen es Sinn machen

würde nur eine Tiefenkarte für Shadow-Mapping und SSAO zu verwenden. Entweder Licht und Kamera sind relativ nahe beieinander wie es in manchen Computerspielen der Fall ist, bei denen der Spieler eine Taschenlampe trägt. Oder es wird ein Außenareal dargestellt bei dem die Shadow-Map für die Sonne berechnet wird und der Spieler hat die Sonne im Rücken, blickt also ungefähr in dieselbe Richtung wie die Lichtquelle.

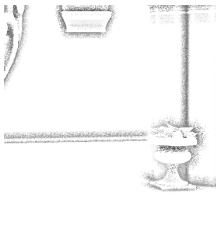
Insgesamt scheint der Aufwand und die resultierenden Probleme unverhältnismäßig hoch, wenn man in Betracht zieht, dass das Erzeugen der Tiefenkarte weniger Zeit in Anspruch nimmt als das Berechnen des SSAO-Wertes. Auch wenn sich das Erstellen der Tiefenkarte aus Sicht der Kamera nicht ersparen lässt, können die Informationen aus einer Tiefenkarte mit einer anderen Perspektive dennoch für die Berechnung von SSAO nützlich sein. Vardis et al. beschreiben wie mehrere Tiefenkarten zusammen zu einem besseren SSAO Ergebnis führen können [VPG13].



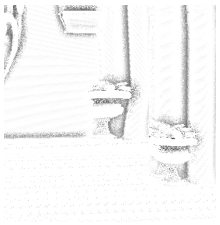
(a) VO mit z-Werten aus der Kamera-Perspektive



(b) VO mit z-Werten aus der Perspektive der Lichtquelle

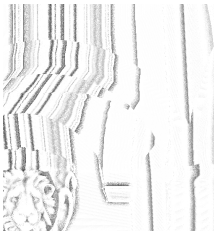


(c) Original



(d) Projektion

(e) Original



(f) Projektion

Abbildung 6.2: Verwenden der z-Werte aus der Lichtquellenperspektive (Bild-6.2b) im Vergleich zur Kameraperspektive (Bild-6.2a). Die Bilder 6.2c und 6.2e sind Ausschnitte aus 6.2a, während 6.2d und 6.2f Ausschnitte aus 6.2b sind.

6.4 Qualität der dynamischen Skalierung

6.4.1 Auswahl der Algorithmen

In Kapitel 4.2.2 wurden 6 verschiedene Möglichkeiten beschreiben, wie der Sample-Radius dynamisch skaliert werden kann. Die ersten 5 Algorithmen

beschränken sich darauf die Amplitude der Tiefenkarte auszuwerten, während der letzte auch die Frequenz mit einbezieht. Da nicht alle Möglichkeiten gute Ergebnisse liefern, wird vor der genaueren Untersuchung eine Vorauswahl getroffen. Die Abbildungen 6.3 bis 6.8 zeigen die Szene Sponza 1, wobei in der linken Bildhälfte der Sample-Radius für VO dynamisch skaliert wurde und, zum Vergleich, in der rechten Bildhälfte VO mit einem festen Radius von 20 berechnet wurde. Die Algorithmen arbeiten auf Basis von 328 Samples, um für die Auswahl möglichst gute Bilder zu verwenden.

Es sind jeweils 3 farbige Rechtecke dargestellt. Beim roten Rechteck gibt es im linken Bereich große Sprünge in der Tiefenkarte, weil hier die Säule aufhört. Im grünen Rechteck sind sowohl feine Strukturen (Blätter innerhalb der Vase), als auch gröbere (der Sockel der Vase) enthalten. Im linken Teil des blauen Rechtecks verkleinern sich die Tiefenwerte an der Säule stetig bis im rechten Teil ein Sprung erfolgt, da hier der Vorhang hinter der Säule zu sehen ist. Die drei Rechtecke zeigen die wichtigsten Größen der Algorithmen: Es sind sowohl große Änderungen in der Amplitude der Tiefenwerte (rotes Rechteck) als auch Änderungen in der Frequenz (grünes Rechteck, Details der Blätter) abgebildet. Des weiteren wird ein schwarzes Rechteck verwendet, um auf Besonderheiten eines Algorithmus hinzuweisen.

Abbildung 6.3 zeigt die Skalierung anhand des maximalen Tiefenwertes in Relation zum Tiefenwert des Punktes. Wegen der großen Änderung in der Tiefe wird im linken Teil des roten Rechtecks der Radius vergrößert, was zu einer unrealistischen Schattenform führt. Im grünen Rechteck gehen Details verloren, da die großen maximalen Tiefenwerte der Säule links dazu führen, dass der Radius vergrößert wird. Der Algorithmus ist ungeeignet, er führt zu unrealistischen Schattenverläufen und lässt in ungünstigen Fällen Details verschwinden.

Abbildung 6.4 skaliert anhand des mittleren Teifenwerts aller Samples die näher am Betrachter sind als der Sample-Punkt. Im linken Teil des roten Bereichs ist wieder ein unrealistischer Schattenverlauf zu sehen, wenn auch nicht ganz so stark wie bei Abbildung 6.3. Im blauen Bereich sieht man, dass der Sample-Radius immer weiter vergrößert wird, je weiter die Sample-Position an der Säule nach hinten geht. Der Durchschnittliche Maximal-Tiefenwert steigt kontinuierlich an und bewirkt einen unrealistischen Schattenverlauf. Im grünen Bereich sind mehr Details sichtbar als bei der Vase im rechten Teil des Bildes, ohne dynamische Skalierung. Wegen den offensichtlich falschen Schattenverläufen (rot, blau) ist dieser Algorithmus ebenfalls ungeeignet.

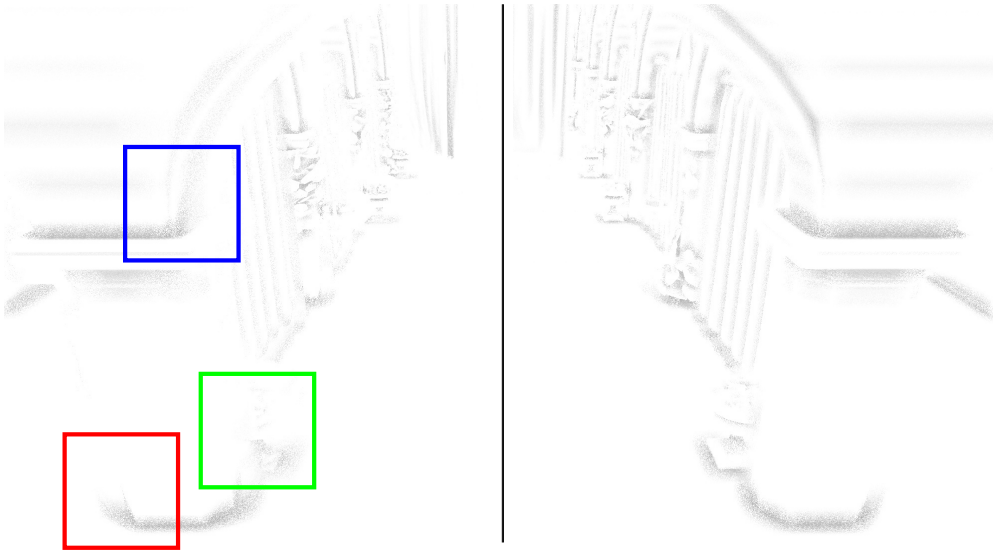


Abbildung 6.3: Linke Bildhälfte dynamischer Algorithmus 1: Maximaler Tiefenwert, rechte Bildhälfte ohne dynamische Skalierung

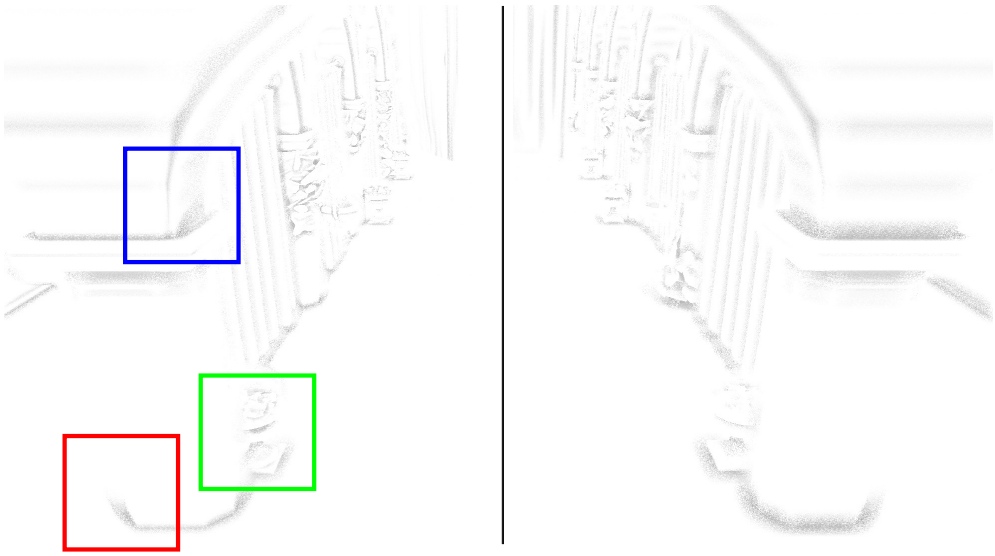


Abbildung 6.4: Linke Bildhälfte dynamischer Algorithmus 2: Mittlerer Tiefenwert, rechte Bildhälfte ohne dynamische Skalierung

Die Skalierung über die Standardabweichung in Abbildung 6.5 verbessert oder verschlechtert das Ergebnis nicht. Der Algorithmus ist daher ungeeignet.

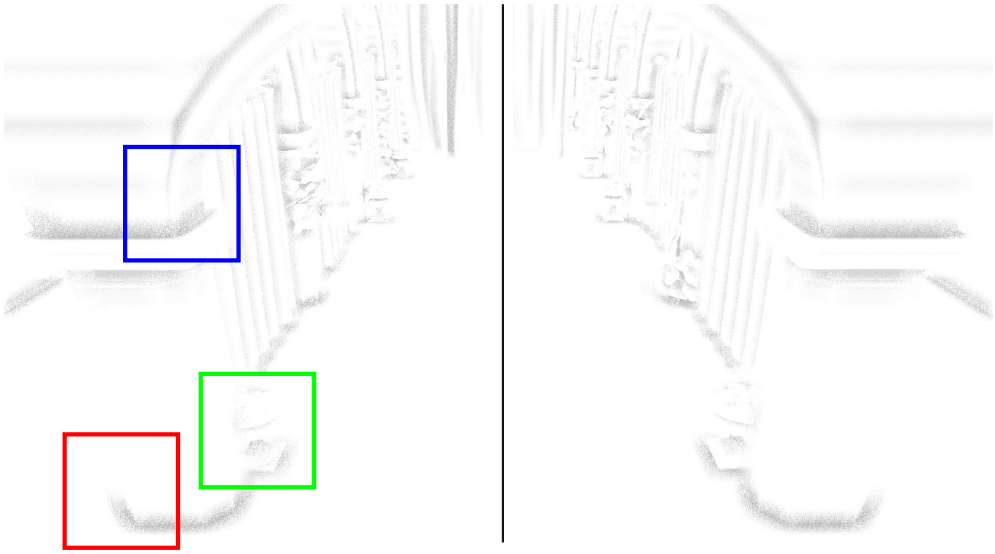


Abbildung 6.5: Linke Bildhälfte dynamischer Algorithmus 3: Standardabweichung, rechte Bildhälfte ohne dynamische Skalierung

Abbildung 6.6 verbessert sich gegenüber 6.3 im roten, blauen und grünen Bereich, weil zu große Sprünge in den Tiefenwerten und damit Sample-Radien gefiltert werden. Im schwarzen Rechteck werden weniger Details angezeigt und es ist ein harter Schnitt zu sehen, an der Stelle, an der die Tiefenwerte die Beschränkung 20 überwinden. Algorithmus Nummer 4 ist am Besten geeignet von den amplitudenbasierten Algorithmen (1-5) und wird daher weiter untersucht.

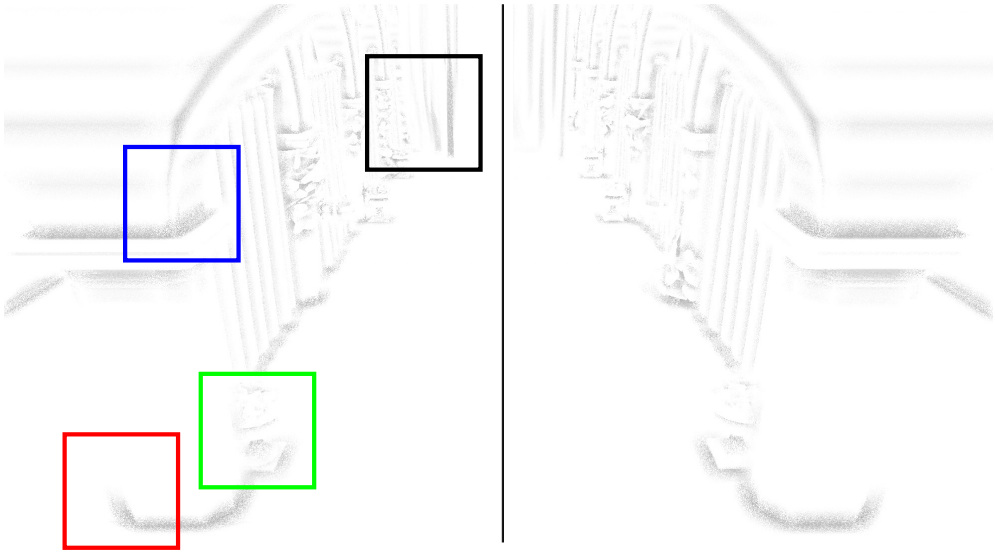


Abbildung 6.6: Linke Bildhälfte dynamischer Algorithmus 4: Maximaler Tiefenwert mit beschränkter Differenz (20), rechte Bildhälfte ohne dynamische Skalierung

Der Algorithmus 5 in Abbildung 6.7 setzt den Sample-Radius auf die Länge des Vektors vom Sample-Punkt zum größten gefundenen Tiefenwert. Die Probleme die dabei auftreten sind dieselben wie bei 6.3, nur dass diese Variante den Vorteil hat, dass der auf diese Weise ermittelte Sample-Radius nicht wie bei Algorithmus 1 mit einem Faktor multipliziert werden muss.

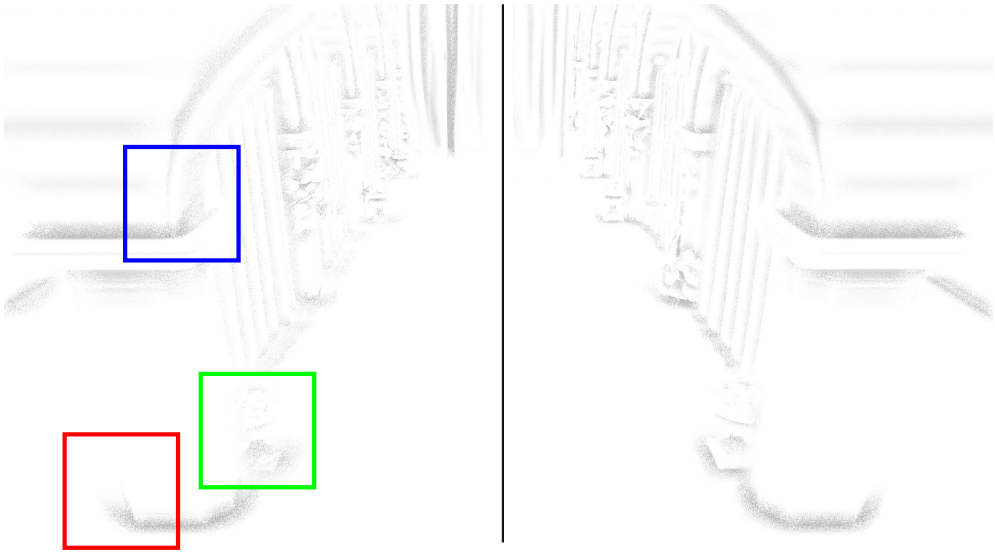


Abbildung 6.7: Linke Bildhälfte dynamischer Algorithmus 5: Vektorlänge zum größten gefundenen Tiefenwert, rechte Bildhälfte ohne dynamische Skalierung

Algorithmus 6 in Abbildung 6.8 versucht die Frequenz der Tiefenkarte mit einzubeziehen. Er sucht den kürzesten Vektor zu einem Minimum innerhalb des Sample-Radius. In hochfrequenten Bereichen des Bildes, wie z.B. im grünen Teil wird der Radius daher verkleinert. Im roten und blauen Bereich bleibt der Sample-Radius gleich, da bei den eher niederfrequenten Bildbereichen das Minimum am Rand des Sample-Radius gefunden wird.

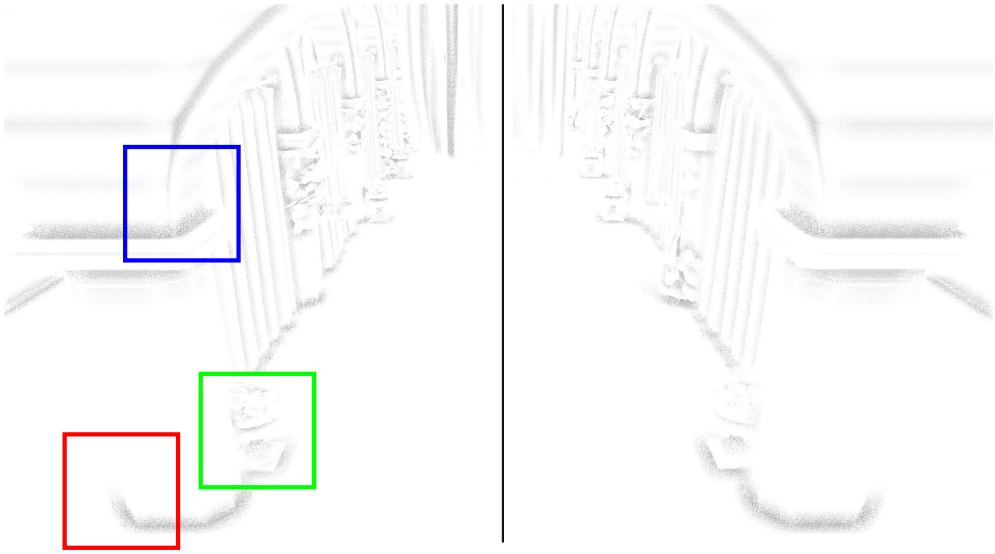


Abbildung 6.8: Linke Bildhälfte dynamischer Algorithmus 6: Vektorlänge des kürzesten Vektors zu einem Minimum, rechte Bildhälfte ohne dynamische Skalierung

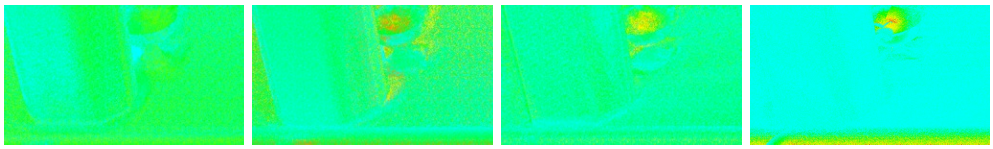
Bei der Betrachtung von Abbildung 6.9 wird klar, dass durch die fixe Grenze von 20 Tiefenwerten starke Artefakte entstehen können. In Abbildung 6.9a sieht man, das der Radius beim Wandteppich bis zu einer gewissen grenze nicht verändert wird, da hier die Differenz aus maximaler Tiefe und Sample-Punkt-Tiefe größer als 20 ist. Ab einem bestimmten Punkt greift dann die Skalierung und es entsteht ein störender Doppelschatten. Im folgenden wird nur noch Algorithmus 6 genauer untersucht, da er am vielversprechensten aussieht.

(a) Algorithmus 4 zeigt einen deutlichen Schnitt bei der maximalen Tiefendifferenz von 20

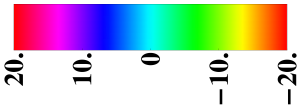
(b) Algorithmus 6 hat keine Probleme mit dieser Situation

Abbildung 6.9: Die Szene Sponza 3 zeigt die Grenzen von Algorithmus 4 auf. Die Beschränkung der Tiefenwerte ist mit 20 in diesem Fall zu groß gesetzt.

(a) 4 Samples (b) 9 Samples (c) 20 Samples (d) 328 Samples



(e) 4 Samples (f) 9 Samples (g) 20 Samples (h) 328 Samples



(i) Skalierung des Sample-Radius relativ zu dem Ausgangsradius 20

Abbildung 6.10: Skalierung des dynamischen Radius anhand 4, 9, 20 und 328 Samples. Die erste Zeile zeigt das Ergebnis der dynamischen Skalierung. Die zweite Zeile zeigt wie sehr sich der Radius durch die Skalierung ändert. Hierzu ist in 6.10i die Farbe abgebildet. Bei Hellblau hat sich der Ausgangsradius von 20 nicht verändert. Bei den gelben Bereichen hat er sich um ca. -10 auf ca. 9 verkleinert.

6.4.2 Sample Anzahl

In Abbildung 6.10 sind Ausschnitte aus der Szene Sponza 1 zu sehen, die jeweils mit einer anderen Anzahl an Samples zum Bestimmen des Vektors erstellt wurden. Vergleicht man die Bilder von 4 bis 328 Samples, dann fällt auf, dass der Radius bei 4 Samples oft als zu klein eingeschätzt wird. Betrachtet man die zugehörigen Skalierungsbilder 6.10e bis 6.10h wird erkennbar, dass der Radius besser eingeschätzt wird, wenn mehr Samples verwendet werden. Bei Abbildung 6.10h sind die meisten Regionen Hellblau, hier hat sich also der Radius nicht verändert. An der Stelle der Vase, an der eine Verkleinerung des Radius gewünscht ist, sind die Farben Grün bis Rot zu erkennen die eine Verkleinerung anzeigen. Diese Farben sind auch bei 9 und 20 Samples zu erkennen.

Insgesammt lässt sich mit Hilfe der Skalierungsbilder erkennen, dass mit zu wenigen Sampels (Abbildung 6.10e) nicht die hochfrequenten Bereiche des Bildes gefunden werden.

6.4.3 Ergebnis

Die Abbildungen 6.11, 6.12 und 6.13 zeigen den dynamischen Radius im Vergleich zu normalem VO. Es ist jeweils zuerst ein Referenzbild[10], darunter ein Vergleichsbild mit dynamischem Radius[11] zu sehen. Die grünen Rechtecke markieren positive Beispiele, die roten negative Beispiele. Darauf folgend ist links ein Differenzbild zwischen den darüber gezeigten Bildern zu sehen. Das Differenzbild zeigt das Negativ der absoluten Differenz der beiden Bilder nach der jeweiligen Anwendung eines Tiefpassfilters mit Radius 15. Schwarz bedeutet also, dass es an dieser Stelle eine Differenz gibt. Daneben ist die Radius-Skalierung visualisiert. Zum Schluss wird der Radius-Skalierung einer Farbskala zugeordnet, die relativ zum Ausgangsradius ist.

Als positives Beispiel ist Abbildung 6.11b zu betrachten. Hier konnten keine visuellen Artefakte, die durch dynamische Skalierung des Sample-Radius entstehen, festgestellt werden. In den grün markierten Bereichen sind neue Details zu erkennen, die in 6.11a nicht so deutlich zu erkennen sind.

In der Sponze-Szene in Abbildung 6.12 werden ebenso mehr Details sichtbar, aber der Radius wird in dem roten Bereich falsch eingeschätzt. Der Schatten wird schräg abgeschwächt, da die hervorstehende Ecke des Sockels als Maximum erkannt wird. In diesem Bereich wird der Schatten verkleinert.

Die Restaurant Szene zeigt, dass nicht mehr Details sichtbar werden. Bei den markierten Stuhlbeinen wird der Radius falsch eingeschätzt. Dies passiert, da am Rand des Image-Space die Tiefenkarte mit dem Textur-Fortsetzungsmodus GL_CLAMP als ebene Fläche fortgesetzt wird. Je näher der Sample-Punkt am Rand ist, desto kleiner wird der Radius eingeschätzt, da das Maximum immer näher an P liegt. Dieser Effekt ist bei allen Skalierungsbildern am unteren Rand zu sehen und tritt immer auf, wenn eine Fläche zu einem der Ränder des Bildes dem Betrachter nähert. Um dies zu verhindern müssten Samples, die außerhalb der Tiefentextur liegen als ungültig verworfen werden. Außerdem wäre es möglich den Viewport zu vergrößern, wie es von Bavoil und Sainz vorgeschlagen wird [LB13a].

In den Skalierungsbildern aller Szenen sind orangefarbene Stellen zu sehen, an denen der Radius klein gewählt wird. Dies geschieht bei einer Geometrie, die eine nach außen gewölbte Oberfläche hat, wie es z.B. bei der Schnauze des Löwen der Fall ist. Hier wird das Maximum an einer Stelle sehr nahe beim Sample-Punkt P gefunden, was zu einem kleinen Radius führt, der gegen 0 geht. Hier könnte der Algorithmus durch vorzeitiges Verlassen optimiert werden, da an derartigen Stellen das SSAO-Ergebnis auf 0% verdeckt gesetzt werden kann.

Andererseits ist derartige Geometrie auch ein potenzieller Bereich, in

[10] VO mit Paired-Samples; ohne Dual-Radii; der Kontrast wurde zur einfacheren Sichtung von Unterschieden erhöht

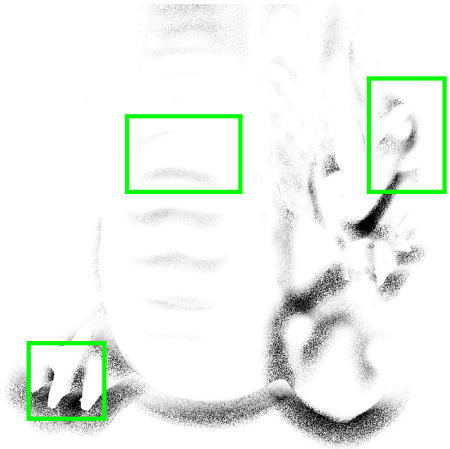
[11] Algorithmus 6; ebenfalls erhöhter Kontrast

dem durch Skalierung des Sample-Radius mehr Details gefunden werden können, denn die im großen Maßstab nach außen gewölbten Oberflächen, die im kleinen weitere Details aufweisen, sind die Stellen, an denen der Radius ohne Verlust von weitläufigen Schatten verkleinert werden kann[12]. Die Schnauze des Löwen weist feine Details im Bereich der Nase auf. Bei einer geringen Sample-Anzahl könnten Details verloren gehen, wenn beispielsweise das Maximum von Nase zu Schnauze nicht gefunden wird und der Algorithmus vorzeitig 0% Verdeckung als Ergebnis liefert.

Insgesammt konnte erfolgreich gezeigt werden, dass durch die dynamische Skalierung des Sample-Radius mehr Details in einem SSAO Durchgang ermittelt werden können. Momentan wird der Sample-Radius lediglich verkleinert. Es bleibt zu prüfen, ob eine Vergrößerung sinnvoll ist und eine weitere Verbesserung des Algorithmus denkbar ist. Das Verfahren ist mit 328 Samples noch unperformant und langsamer als das Dual-Radii Verfahren von Loos und Sloan, aber es ist zu erkennen, dass es sich lohnen würde weiter in diese Richtung zu forschen.

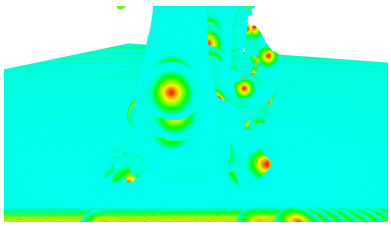
[12] Siehe Kapitel 4.2.1 mit dem Ringing-Beispiel aus Tabelle 4.1a

(a) Referenz VO mit einem Radius von 4,8

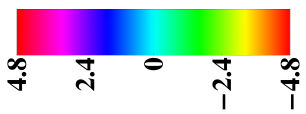


(b) VO mit dynamischem Radius mit 328 Samplen

(c) Differenzbild zwischen 6.11a und 6.11b

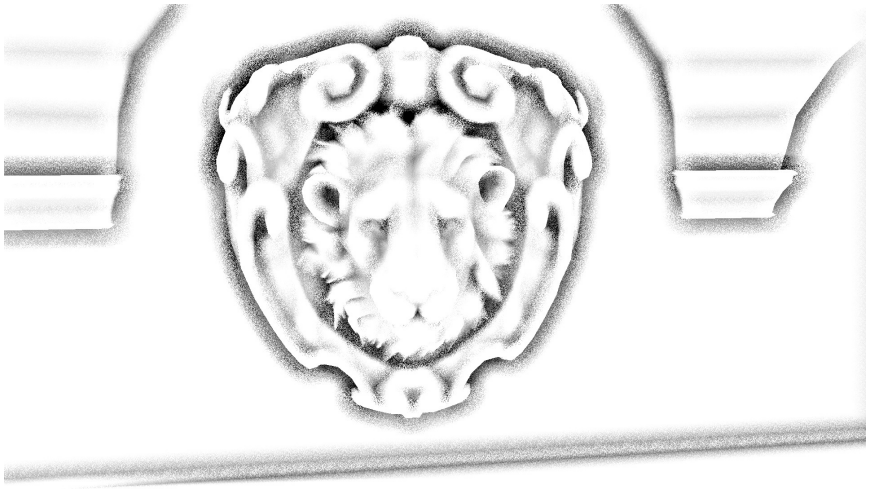


(d) Skalierungsbild

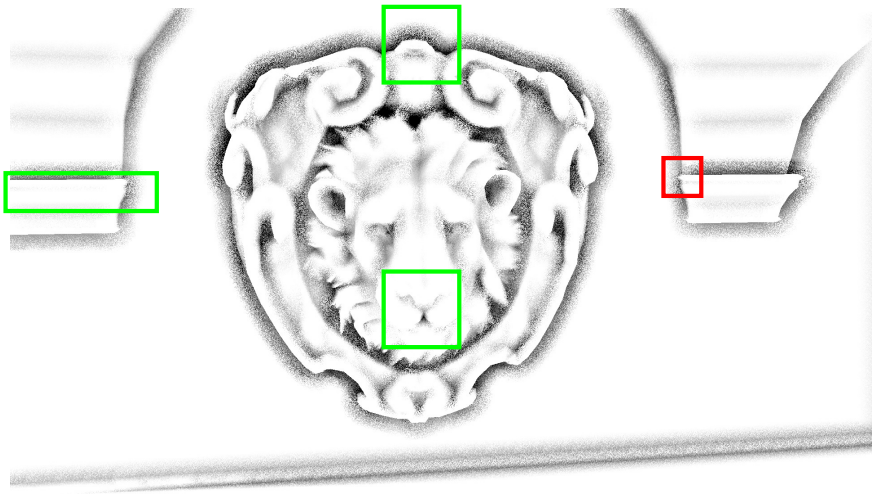


(e) Farbzuordnung zu Radiusdifferenz von 6.11d. Hellblau bedeutet der Radius ist unverändert

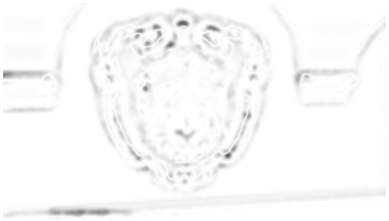
Abbildung 6.11: Szene: Stanford Drache



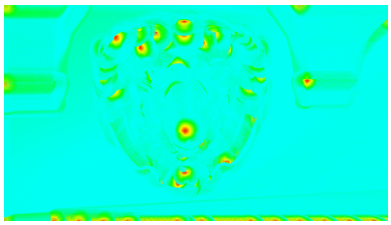
(a) Referenz VO mit einem Radius von 20



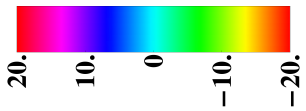
(b) VO mit dynamischem Radius mit 328 Samplen



(c) Differenzbild zwischen 6.12a und 6.12b



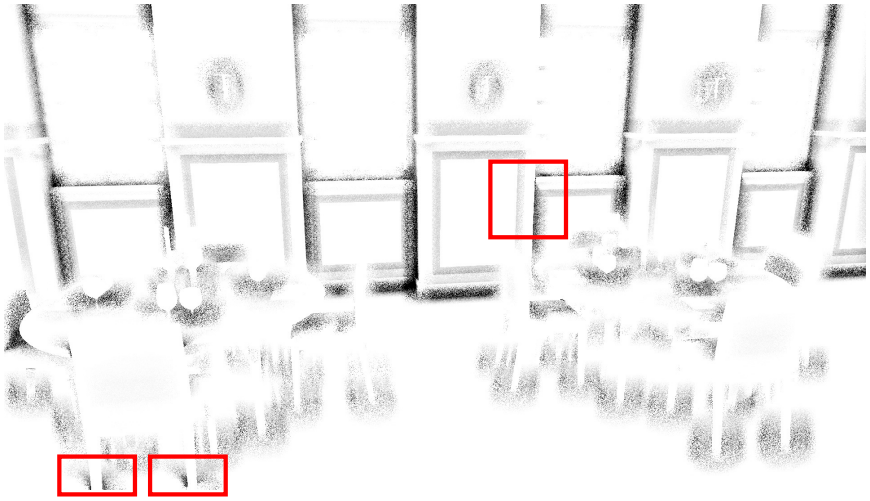
(d) Skalierungsbild



(e) Farbzuordnung zu Radiusdifferenz von 6.12d. Hellblau bedeutet der Radius ist unverändert

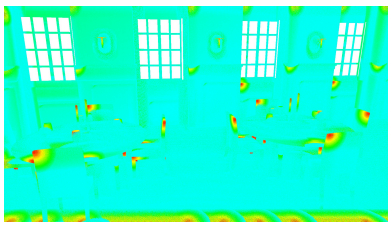
Abbildung 6.12: Szene: Sponza 2

(a) Referenz VO mit einem Radius von 0.12

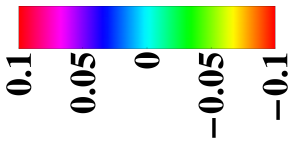


(b) VO mit dynamischem Radius mit 328 Samplen

(c) Differenzbild zwischen 6.13a und 6.13b



(d) Skalierungsbild



(e) Farbzuordnung zu Radiusdifferenz von 6.13d. Hellblau bedeutet der Radius ist unverändert

Abbildung 6.13: Szene: Restaurant

7 Fazit und Ausblick

7.1 Fazit

Ein Ziel dieser Arbeit war es, aktuelle SSAO Algorithmen anhand visueller Qualität und Performanz zu vergleichen. Zu diesem Zweck wurden die Algorithmen SSAO, VO und HBAO in einem OpenGL Programm implementiert. Es konnte gezeigt werden, dass HBAO und VO im Vergleich zu SSAO eine bessere Qualität erreichen. HBAO und VO nutzen die Informationen aus den einzelnen Samples besser aus, während bei SSAO lediglich festgestellt wird, ob ein Sample von Geometrie verdeckt wird oder nicht. Außerdem ist die Sample-Invalidierung bei SSAO nicht optimal gelöst, da sie von der Distanz zum Betrachter abhängig ist. VO löst dies besser mittels einer maximalen Occlusion-Distanz im View-Space. Durch die Verwendung von Sample-Paaren können Samples mit ungültiger Distanz als eine ebene Fläche approximiert werden. Bei HBAO ist die maximale Occlusion-Distanz ebenfalls im View-Space, durch den Sample-Radius gegeben. Eine Verwendung von Sample-Paaren ist durch das Ray-Marching nicht nötig. Samples die außerhalb der Hemisphäre liegen werden ignoriert und vergrößern den Horizontwinkel nicht. HBAO bietet außerdem einen Angle-Bias, um Schattierung durch niedrige Tesselierung von Geometrie zu verhindern. Auch die Anzahl der Samples lässt sich über Parameter bestimmen. HBAO ist am flexibelsten und kann für eine große Bandbreite von Hardware konfiguriert werden, während die Stärken von VO in der besseren Performanz bei wenigen Samples liegen.

Ein weiteres Ziel dieser Arbeit war es die Qualität oder Performance von SSAO Algorithmen zu verbessern. Im Bezug zur Performance wurde geprüft, ob es sinnvoll ist die Tiefenkarte aus Perspektive der Kamera nicht zu erstellen und stattdessen die Tiefenkarte aus Sicht einer Lichtquelle, wie sie beim Shadow-Mapping erzeugt wird, zu verwenden. Es hat sich gezeigt, dass dies nur unter speziellen Voraussetzungen funktioniert. Die Lichtquelle und Kamera müssen in die selbe Richtung blicken und das View-Frustum der Lichtquelle muss das der Kamera komplett beinhalten. Sehen Kamera und Lichtquelle von der selben Position in die gleiche Richtung, sind die direkten Schatten, die beim Shadow-Mapping berechnet werden, nicht sichtbar. In der Praxis sind sie am deutlichsten zu sehen, wenn die Lichtquelle in eine andere Richtung blickt als der Betrachter. Da das Erstellen der Tiefenkarte wesentlich weniger Zeit in Anspruch nimmt, als den Occlusion-Wert zu berechnen und sich aus der Verwendung einer anderen Tiefenkarte Probleme ergeben, scheint es wenig sinnvoll auf die Tiefenkarte in der Kameraperspektive gänzlich zu verzichten oder zwischen Fällen zu unterscheiden, bei denen es sich lohnen bzw. nicht lohnen würde.

In Bezug zur visuellen Qualität konnte am Beispiel von VO erfolgreich gezeigt werden, dass sich durch die dynamische Skalierung des Sample-Radius

mehr Details aus der Tiefenkarte ermitteln lassen. Dabei wurde VO bewusst gewählt, da hier die Qualität mehr vom gewählten Sample-Radius abhängt als es bei HBAO, durch die Verwendung des Horizontwinkels, der Fall ist. Zur dynamischen Skalierung wurden mehrere Algorithmen erarbeitet, die die Tiefenkarte vor der VO-Berechnung sondieren. Es hat sich gezeigt, dass es nicht ausreicht, den Radius anhand der Amplitude in der Tiefenkarte zu skalieren. Es muss auch die Frequenz betrachtet werden. Bei der selben Amplitude muss der Radius bei einer hohen Frequenz kleiner gewählt werden, als es bei einer niedrigeren Frequenz der Fall wäre. Die Algorithmen, die nur die Amplitude auswerten brachten keine guten visuellen Ergebnisse. Erst durch Beachtung der Frequenz, durch die Suche des Maximums mit der kürzesten Entfernung zum Sample-Punkt, konnte die Qualität verbessert werden. Es wurde gezeigt, dass der Algorithmus bei einer geringen Sample-Anzahl dazu tendiert, den Radius zu klein zu wählen oder Details übersehen werden. Dies geschieht, da das korrekte Maximum mit einer geringen Sample-Anzahl nicht gefunden wird.

7.2 Ausblick

Beim Vergleich der Algorithmen musste darauf geachtet werden, die selbe Anzahl an Samples bei allen Algorithmen zu verwenden. Loos und Sloan [LS10] erzielen jedoch bereits mit 5 Line-Samples ein qualitativ akzeptables Ergebnis. Es würde sich lohnen zu überprüfen, wie die Performanz und visuelle Qualität von VO bei einer geringeren Anzahl an Line-Samples ist, als es in dieser Arbeit erfolgte. Die Qualität und Performanz von VO könnte durch bessere Vorbereitung der Samples noch weiter gesteigert werden. Es könnte auch verglichen werden, inwiefern sich eine schlechte von einer optimalen Sampleverteilung im Ergebnis unterscheidet.

Des weiteren ermöglicht derzeit lediglich die Implementierung von HBAO die Benutzung von Normalenvektoren für die Berechnung des Occlusion-Werts. Sowohl SSAO als auch VO können von der Verwendung von Normalen profitieren [Mit07, LS10]. Es könnte überprüft werden, wie weit die Qualität gesteigert werden könnte und wie stark die Performanz beeinflusst würde.

Außerdem wurde bisher auf die Implementierung eines speziellen Tiefpassfilters verzichtet, der nicht über Kanten in der Tiefenkarte hinweg glättet. Ein derartiger Filter würde das Rauschen, dass durch die Erzeugung von mehr Sample-Richtungen entsteht, entfernen. Sowohl SSAO, VO als auch HBAO werden gefiltert um das finale Ergebnis zu erhalten[Mit07, LS10, BSD09].

Die dynamische Skalierung ist in der derzeitigen Implementierung mit 328 Samples nicht performant und kann weiter optimiert werden. Mit einer optimierten Version könnten geklärt werden, ob die Performanz und die Qualität vergleichbar mit dem Dual-Radii verfahren von Loos und Slo-

an ist. Ebenso könnte überprüft werden, wie sehr die Qualität des SSAO-Algorithmus von einer dynamsichen Skalierung des Radius profitieren würde.

Literatur

[AMHH11] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering, Third Edition*. Taylor & Francis, 2011.

[BSD09] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *Shaderx7* [eng09], pages 425–444.

[Chr03] P.H. Christensen. Global illumination and all that. In *ACM SIGGRAPH Course 9*, volume 9, 2003.

[DG13a] Rob Knapp Darren Glosemeyer. Random Number Generation, March 2013. URL: `http://www.wolfram.com/learningcenter/tutorialcollection/RandomNumberGeneration/RandomNumberGeneration.pdf`.

[DG13b] Rob Knapp Darren Glosemeyer. Sphere Point Picking, March 2013. URL: `http://mathworld.wolfram.com/SpherePointPicking.html`.

[eng09] *Shaderx7*, Shaderx series. Charles River Media, 2009.

[ESAW11] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-Time Shadows*. A.K. Peters, 2011.

[Fir09] Michael Firbach. Screen Space Ambient Occlusion, June 2009.

[Isi13] John R. Isidoro. Shadow Mapping: GPU-based Tips and Techniques, March 2013. URL: `http://developer.amd.com/wordpress/media/2012/10/Isidoro-ShadowMapping.pdf`.

[JPO10] Christopher Hall John-Paul Ownby, Robert Hall. Toy Story 3: The Video Game Rendering Techniques. In *Advances in Real-Time Rendering, SIGGRAPH 2010 courses, ACM*, 2010.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[Kaj09] Vladimir Kajalin. Screen-space ambient occlusion. In *Shaderx7* [eng09], pages 413–424.

[Lan02] Hayden Landis. Production-Ready Global Illumination. In *Siggraph Course Notes*, volume 16, 2002.

[LB13a] Miguel Sainz Louis Bavoil. Multilayer SSAO, March 2013. URL: http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/Bavoil_MultiLayerDualResolutionSSAO.pdf.

[LB13b] Miguel Sainz Louis Bavoil. Screen Space Ambient Occlusion Example Project, February 2013. URL: http://developer.download.nvidia.com/SDK/10.5/Samples/ScreenSpaceAO.zip.

[LS10] Bradford James Loos and Peter-Pike J. Sloan. Volumetric obscurance. In Daniel G. Aliaga, Manuel M. Oliveira, Amitabh Varshney, and Chris Wyman, editors, *SI3D*, pages 151–156. ACM, 2010.

[Mit07] Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[NFHS11] A. Nischwitz, M. Fischer, P. Haberäcker, and G. Socher. *Computergrafik und Bildverarbeitung: Band I: Computergrafik*. Vieweg + Teubner Studium. Vieweg+Teubner Verlag, 2011.

[VPG13] Kostas Vardis, Georgios Papaioannou, and Athanasios Gaitatzes. Multi-view ambient occlusion with importance sampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 111–118, New York, NY, USA, 2013. ACM.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson L. Max, editors, *Rendering Techniques*, pages 45–56. Springer, 1998.