Karlsruhe Institute of Technology

Schattenverfahren auf der GPU

Proseminar-Ausarbeitung von

Karl Sassie

An der Fakultät für Informatik
Institut für Visualisierung und Datenanalyse,
Lehrstuhl für Computergrafik

30. November 2022

Betreuender Mitarbeiter: Vincent Schüßler

Abstract

Die Computer Grafik hat in den letzten Jahren sich sehr weit weiter entwickelt und was vor 50 Jahren noch undenkbar war, kann heute in Echtzeit berechnet werden. Dahinter verbirgt sich einerseits die Weiterentwicklung der Hardware, doch auch im Software Bereich sind große Forschungssprünge zu vermerken. Ein wichtiger Teil der Computer Grafik ist die Darstellung von Schatten. Deshalb will ich mich in dieser Ausarbeitung mit Algorithmen auseinandersetzen, die nicht nur eine gute Qualität haben, sondern auch in Echtzeit berechnet werden können. Dabei werde ich mich auf eine Technik fokussieren, die 1978 von Williams vorgestellt wurde, die „Shadow Mapping“ heißt. Diese ist in der einfachsten Form leicht zu verstehen, doch es gibt viele Erweiterungen des Verfahrens die, die visuelle Qualität der Schatten verbessern. Die Erweiterungen des „Shadow Mapping“ Verfahrens beheben meist ein visuelles Artefakt. Die Artefakte vollständig zu verstehen macht es leichter die einzelnen Erweiterungen zu nachzuvollziehen. Deshalb will ich diese erläutern und anschließend Anpassungen des Shadow Mapping Verfahrens vorstellen.

Inhaltsverzeichnis

1. Einleitung

Warum sind Schatten in der Computer Grafik eigentlich so wichtig? Schatten geben nicht nur besser aussehende Bilder, sondern geben auch wichtige Informationen über Position und Skalierung von Objekte (Abbildung 1.1).

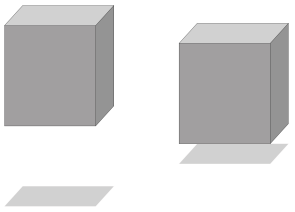

Abbildung 1.1: Es ist klar zu erkennen, das der linke Würfel höher über dem Boden schwebt als der rechte Würfel. Ohne Schatten wäre es unmöglich zu entscheiden, welcher Würfel wieweit über dem Boden schwebt.

Ein Pixel liegt genau dann im Licht, wenn die Gerade zwischen Pixel und Lichtquelle von keinem anderen Objekte blockiert wird. In einem Raytracing Renderer ist dies leicht umzusetzen. Um zu bestimmen, ob ein Punkt im Schatten liegt kann ein weiterer Ray von dem Oberflächenpunkt zu dem Licht berechnet werden, falls dieser ein Objekt schneidet so liegt der Punkt im Schatten. Raytracing ist jedoch ein sehr aufwendiges Verfahren, deshalb wird in Echtzeitberechnungen von Bildern meistens ein Rasterisierter Renderer verwendet. In einem Rasterisierten Renderer ist es jedoch nicht möglich den Schnitt eines Rays mit der Szene zu berechnen um zu bestimmen, ob ein Punkt im Schatten liegt.

Der bekannteste Algorithmus der Schatten in mit einem Rasterisiertem Renderer berechnet heißt Shadow Mapping. In einem Raytracing Renderer wird für jeden Punkt berechnet, ob das Licht sichtbar ist. Bei Shadow Mapping wird dies umformuliert. Ein Punkt liegt im Schatten, genau dann wenn der Punkt vom Licht sichtbar ist. Dieses Verfahren kommt jedoch mit einigen Problemen, denn um zu berechnen, ob ein Punkt von dem Licht sichtbar ist, wird die Szene aus der Perspektive des Lichtes auf eine Textur gezeichnet. Dabei wird sich für jeden Pixel der Textur gespeichert wie, weit der nächstliegende Punkt von dem Licht entfernt ist. Da es bei dem abspeichern zu einer Diskretisierung kommt rechnen wir in den Folgeschritten niemals mit der tatsächlichen Tiefe eines Punktes, sondern mit einer

Rundung zu dem nächstliegenden Diskretisierungswert. Außerdem ist es durch die endliche Auflösung der Textur nicht möglich die Tiefenwerte eines exakten Punktes zu speichern, sondern nur die Tiefenwerte eines Bereichs, der von einem Pixel auf der Textur beschrieben wird.

Trotz der vielen Probleme wird Shadow Mapping in vielen Echtzeit Applikationen verwendet. Deshalb werde ich mit im weiteren mit dem Algorithmus auseinander setzen und dabei die Probleme die dabei auftreten untersuchen. Dabei werde ich zunächst auf die Wert Diskretisierung in der Tiefen Textur eingehen. Dies führt zu einem Artefakt, dass Selbstverschattung genannt wird. Um dieses Artefakt zu verhindern gibt es viele Möglichkeiten, ich will mich jedoch mit Heuristischen Biasing Methoden und Second Depth Shadow Mapping beschäftigen. Anschließend werde ich auf Projektives und Perspektivisches Aliasing eingehen, diese Artefakte entstehen durch die limitierte Textur Auflösung der Shadow Map. Um diese Artefakte zu beheben, gibt es eine überzahl an Verfahren, ich werde mich jedoch auf PSM, LiSPSM, CSM und PCF beschränken diese Verfahren bieten eine gute Grundlage für das Verständnis weiterer komplexerer Verfahren. Am Ende will ich in einem Fazit eine Echtzeitfähige Kombination der vorgestellten Verfahren vorstellen.

2. Grundlagen der Rasterisierung

2.1 Koordinatensysteme

Bei dem Rasterisierten Rendern zeichnen wir Objekte die aus Oberflächen (Dreiecken) bestehen. Oberflächen bestehen aus Punkten die, die Position der Oberfläche relativ zu dem Objekt Mittelpunkt beschreiben. Diese Koordinaten, die relativ zu dem Objekt geben sind, sind in einem Koordinatensystem, das **Object Space** heißt. Um ein Objekt in der Welt verschieben zu können müssen alle Punkte eines Objekts (in Object Space) mit einer Linearen Transformation in **World Space** transformiert werden. Die Transformation besteht aus Skalierung, Translation und Rotation und wird meistens als 4x4 Matrix mit homogenen Koordinaten dargestellt. Das Konzept einer Kamera lässt sich analog einführen. Dabei werden alle Punkte im World Space mit einer Linearen Transformtion in **View Space** transformiert. In View Space sind alle Punkte relativ zu der Kamera. In einer weiteren Transformation wird nun die Art der Projektion der Kamera definiert. Dabei wird ein Punkt im View Space in **Clip Space** transformiert. Die Transformation transformiert alle Punkte die sichtbar sind auf einen Punkt innerhalb eines Würfels der Größe 2x2x2 um den Ursprung. Alle Oberflächen die außerhalb des Würfels liegen sind von der Kamera nicht sichtbar und werden nicht gezeichnet. Aktuell haben wir nur Transformationen der Form $\mathbb{R}^3 \to \mathbb{R}^3$ betrachtet. Um die Punkte auf dem Bildschirm darzustellen müssen wir eine Transformation der Form $\mathbb{R}^3 \to [0, 1080] \times [0, 1920]$ (Unter der Annahme, das der Bildschirm FullHD ist). Diese Transformation wird von der Renderpipeline implizit durchgeführt, dabei werden alle Punkte in Clip Space auf die XY-Ebene projiziert und die Z-Werte werden als Tiefen Wert verwendet. Die Transformation transformiert somit Punkte von Clip Space in **Screen Space**.

2.2 Projektion

In Abschnitt 2.1 wurde bereits auf die einzelnen Koordinatensysteme und Transformationen eingegangen, für die Kamera Projektion ist jedoch eine genaueres Verständis nötig. Eine Kamera Projektion ist eine Abbildung von View Space in Clip Space. Im Allgemeinen lassen sich alle Kamera Projektionen mit einem Frustum (Kegelstumpf) eindeutig charakterisieren. Ein Kamera Frustum beschreibt einen Raum im View Space, der durch die Projektion auf einen Würfel im Clip Space abgebildet wird. Eine Oberfläche des Frustums, die Near Plane, ist die Oberfläche auf die projiziert wird. Die entgegengesetzte Oberfläche heißt Far Plane. Die Far Plane beschreibt die Punkte, die am weit möglichsten

von der Kamera entfernt sind und noch sichtbar sind. Insbesondere wird eine Gerade von einem Punkt auf der Near Plane zu einem entsprechenden Punkt auf der Far Plane durch die Projektion auf eine Gerade projiziert die orthogonal zu der XY-Ebene liegt. Dadurch wird klar, dass die Projektionstransformation, die Art der Projektion der Kamera definiert.

Projektionstransformationen werden oft mit einer 4x4 Matrix implementiert. Für eine orthogonale Projektion ist dies nicht nötig, da hier von $\mathbb{R}^3$ nach $\mathbb{R}^3$ abgebildet wird, doch bei perspektivischen Projektionen wird die w-Komponente verwendet um die perspektive zu implementieren. Dabei werden nach der Matrix Multiplikation die xyz-Komponenten durch die w-Komponente des Ergebnisvektors geteilt. Dabei ist zu bemerkten, dass die z-Koordinate ebenfalls durch die w-Komponente geteilt wird. Dies ist im allgemeinen sinnvoll, den es führt zu einer nicht linearen Verteilung der Tiefe, wodurch die Tiefenauflösung nahe bei der Kamera höher ist.

2.3 Framebuffer

Bei dem Zeichnen der Szene werden die Farb und Tiefenwerte der Pixel in einem Framebuffer gespeichert. Dadurch das die Werte diskret gespeichert werden, enthält der Framebuffer nach dem rendern nicht die tatsächlichen Farb und Tiefenwerte, sondern eine Rundung auf den nächst gelegenen Werte. Für die Farbwerte ist dieses Verhalten kein Problem, da der Unterschied zwischen 2 Farbwerten, die nebeneinander liegen, nicht erkennbar ist. Dieses Verhalten führt jedoch bei Tiefenwerten dazu, dass die Tiefenwerte des Framebuffers niemals die tatsächliche Tiefe beschreiben. Dieses Problem ist im Allgemeinen durch die nicht lineare Verteilung der Tiefenwerte, zwischen Near und Far Plane, kein Problem, doch im weiteren werden wir sehen, dass dies zu Problemen führt, wenn die Szene aus einer anderen Perspektive gezeichnet wird und die Ergebnisse in einem weiteren Zeichenschritt verwendet werden.

3. Shadow Mapping

3.1 Uniform Shadow Mapping

Uniform Shadow Mapping [Wil78] ist ein Algorithmus zu Berechnung von Schatten mit einem Rasterisiertem Renderer. Es ist das Verfahren, das in der Echtzeitberechnung am meisten zum Einsatz kommt, da es in einem Rasterisiertem Renderer implementierbar ist.

Wie bereits in der Abschnitt 1 erwähnt ist zur Berechnung von Schatten nötig, festzustellen, ob ein anderes Objekt das Licht an einem Punkt blockiert. Diese Berechnung für jeden Pixel direkt auszuführen ist in einem Rasterisiertem Renderer nicht möglich. Beim Shadow Mapping Verfahren formulieren wir unsere Frage um und betrachten die Äquivalente Frage: **Ist der Punkt von der Perspektive des Lichtes Sichtbar?** Wenn ja dann wird das Licht von keinem Objekte blockiert. Dieses umformulieren der Fragestellung hat den Vorteil das sie sich in einem Rasterisierten Renderer berechnen lässt, indem die Szene aus der Perspektive des Lichtes gezeichnet wird.

Der Shadow Mapping Algorithmus lässt sich in 2 Schritte unterteilen:

1. Im ersten Schritt wird ein Koordinatensystem (**Light View Space**) konstruiert, dass die Szene relativ zu dem Licht beschreibt. Anschließend wird auf dieses Koordinatensystem eine Projektionstransformtion angewendet. Das neue Koordinatensystem heißt **Light Clip Space**. Die Szene wird nun mit dem Light Clip Space Koordinatensystem gezeichnet. Dabei muss nur der Tiefenwert für jeden Pixel gespeichert werden. Die Textur mit den Tiefenwerten heißt **Shadow Map**.

2. Im zweiten Schritt wird die Szene auf der Perspektive der Kamera gezeichnet. Gleichzeitig zu der Transformation der Punkte von Object Space zu Clip Space werden die Punkte ebenfalls in Light Clip Space transformiert. Nun kann der Abstand des Punktes zu dem Licht aus der z-Komponente des Punktes im Light Clip Space Koordinatensystems abgelesen werden. Mit der x und y Komponente kann bestimmt werden, auf welchen Pixel in der Shadow Map der Punkt in Schritt 1 abgebildet wurde. Anschließend kann durch einen Vergleich bestimmt werden, ob der Punkt im Schatten liegt. Wenn der Tiefenwert des Pixels in Shadow Map kleiner ist als die z-Komponente des Punktes im Light Clip Space Koordinatensystem, so ist der Punkt von der Perspektive des Lichtes nicht sichtbar und liegt somit im Schatten.

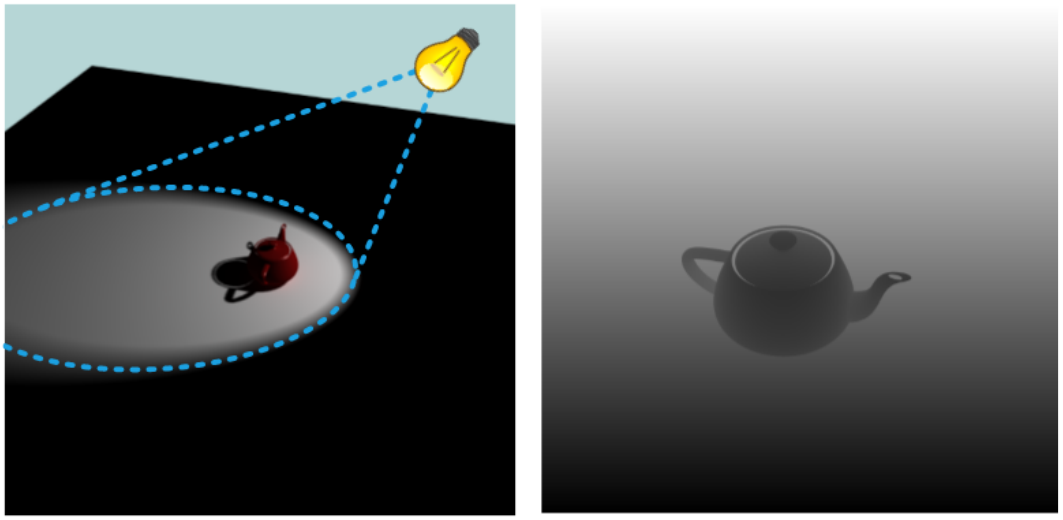

Abbildung 3.1: [Links] die Perspektive der Kamera mit eingezeichneter Lichtposition. [Rechts] Die Tiefenwerte der Szene aus der Perspektive des Lichtes. Diese Textur wird Shadow Map genannt.

3.2 Arten von Lichtquellen

Es gibt unterschiedliche Arten von Lichtern die sich alle mit Shadow Mapping darstellen lassen. Das direktionales Licht ist ein Licht, bei dem alle Lichtstrahlen parallel verlaufen. Dies lässt sich beim Shadow Mapping umsetzen, indem als Projektionsmatrix des Lichtes eine Orthogonale Projektion verwendet wird. Analog lässt sich ein Spot Light implementieren, indem als Projektion eine Perspektivische Projektion verwendet. Punkt Lichter können etwas komplizierter sein, doch wir wollen Punkt Lichter hier einfach als 6 Spotlights betrachten die jeweils 90° abdecken dabei wird die Shadow Map in einer Würfel Textur mit 6 Seiten gespeichert. Der Shadow Mapping Algorithmus verläuft für alle Lichter identisch nur die Wahl der Projektion wird verändert.

3.3 Überblick der Diskretisierung Probleme

Beinahe alle Probleme des Shadow Mapping Verfahren entstehen durch die Diskretisierung, die gezwungenermaßen in Computern auftritt. Dabei tritt Diskretisierung in mehreren Dimensionen auf.

Die erste Form der Diskretisierung die zu Fehlern führt ist die Werte-Diskretisierung in der Shadow Map. Bei dem Vergleich, ob ein Pixel im Schatten liegt oder von einer Lichtquelle beleuchtet wird, kommt es zur falschen Klassifikation. Dieser Fehler entsteht durch runden der Tiefenwerte in der Shadow Map und durch die numerische Instabilität der Lichttransformation. Dies führt zu einem deutlich sichtbaren visuellen Artefakt, das Selbestverschattung genannt wird. Selbestverschattung wird in Abschnitt 3.4 genauer erklärt.

Die zweite Dimension, in der Diskretisierungsfehler auftreten, ist die Pixel Auflösung der Shadow Map. Diskretisierungsfehler die dadurch auftreten erzeugen Aliasing Artefakte. Eine Möglichkeit diese zu verbessern funktioniert analog zu Geometrie Antialiasing, dabei kann jedoch nicht die Shadow Map selbst gefiltert werden, sondern es werden die Vergleichsergebnisse gefiltert. Dies führt dazu das die Schattenkanten geglättet werden. Eine mögliche Implementierung hiervon heißt Percentage Closer Filtering und wird in Abschnitt 3.5 behandelt.

Um andere Methoden zur Verbesserung des Aliasing Problems zu betrachten ist eine genauere Klassifikation des Aliasing Problems nötig. Dabei wird Aliasing in 2 Kategorien eingeteilt:

- **Perspektive Aliasing** entsteht dadurch, dass Oberflächen die nahe bei der Kamera sind groß auf dem Bildschirm dargestellt werden, jedoch wird für die Schattenberechnung dieselbe Anzahl an Shadow Map Pixeln verwendet, wie für Oberflächen, die sehr weit von der Kamera entfernt sind. Der Effekt von Perspektive Aliasing ist maximal, wenn das Licht direkt in die Kamera leuchtet, denn hier sind Oberfläche nahe bei der Kamera im Clip Space groß aber im Light Clip Space klein, wodurch Oberflächen nahe bei der Kamera wenig Pixel auf der Shadow Map zugeordnet sind. Dieses Szenario nennt man Duelling Frusta. Analog ist der Effekt minimal, wenn die Lichtrichtung gleich der Sichtrichtung ist.

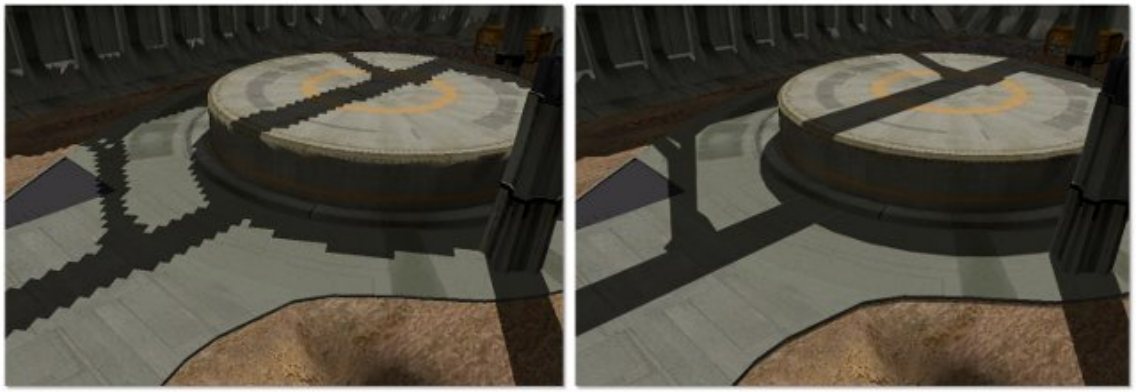

Abbildung 3.2:

- **Projektives Aliasing** entsteht, wenn zwischen Pixeln in der Shadow Map und Pixeln im Screen Space kein 1:1 Verhältnis existiert. Dieser Effekt ist besonders stark bei Oberflächen mit Normalen orthogonal zu der Lichtrichtung. Diese Oberflächen nehmen auf der Shadow Map nur sehr wenig Pixel ein, wobei sie im Screen Space sehr groß sein können.

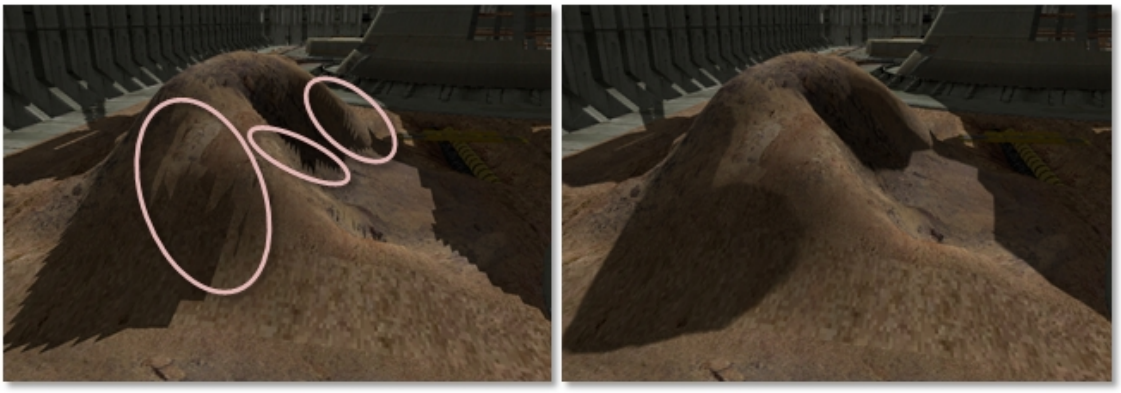

Abbildung 3.3:

3.4 Selbestverschattung

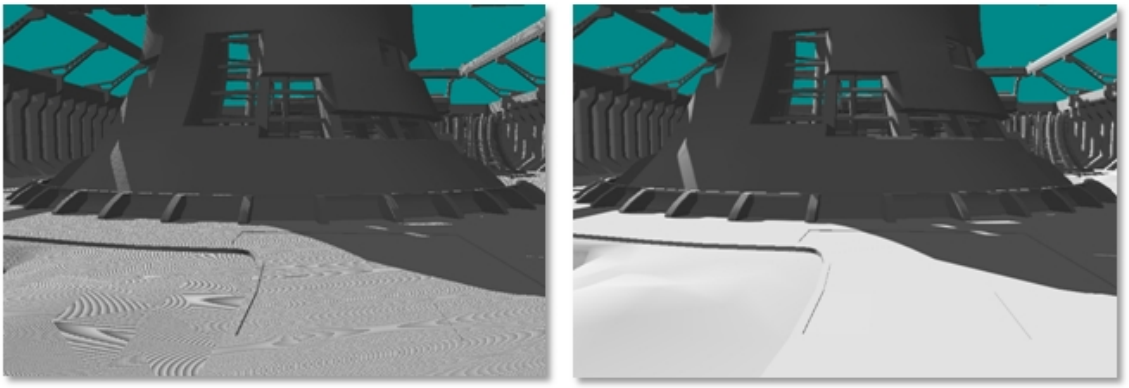

Abbildung 3.4: Links : Szene mit Selbstverschattung.
Rechts: Szene ohne Selbstverschattung.

Selbstverschattung (Abbildung: 3.4) ist ein visuelles Artefakt. Es tritt auf, wenn ein Punkt in der Shadow Map näher ist als tatsächlich, wodurch der Punkt sich selbst beschattet. Dies kann an folgen Gründen liegen.

1. Die Rundung der Tiefenwerte kann zu Selbestverschattung führen, dabei kann entweder der Tiefenwert in der Shadow Map abgerundet oder die Z-Komponente des Punktes in Light Clip Space aufgerundet worden sein.
2. In der Shadow Map kann nicht die tatsächliche Tiefe eines Punktes gespeichert werden, sondern nur die Tiefe eines Bereichs, der von einem Pixel in der Shadow Map beschrieben wird. Für Oberflächen die parallel zur Lichtrichtung liegen und klein auf der Shadow Map dargestellt werden ist dieses Problem verstärkt.

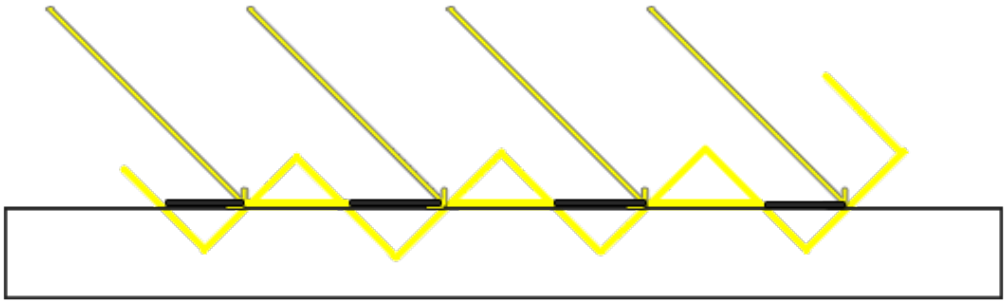

Abbildung 3.5: Die Box ist ein beleuchtetes Objekt. Die zackige gelbe Linie beschreibt die Tiefenwerte die tatsächlich in der Shadow Map gespeichert sind. Hier sind 2 Diskretisierungen zu bemerken; die Breite der Shadow Map Pixel und der Höhenunterschied der Pixel (Tiefenauflösung). Der schwarze Teil der Box Oberfläche liegt im Licht, da hier der Tiefenwert in der Shadow Map größer ist als der Abstand der Oberfläche zu dem Licht. analog liegt der gelbe Teil der Box Oberfläche im Schatten.

Selbstverschattung kann durch einen Bias-Wert verringert werden. Dabei wird vor dem Vergleich ein kleiner Wert (der Bias), auf den Tiefenwert in der Shadow Map addiert.

Bildlich kann man sich vorstellen, dass die gelbe gezackte Linie (aus Abbildung: 3.5) in die Lichtrichtung verschoben wird. Es tritt keine Selbstverschattung mehr auf, wenn die komplette gelbe Linie unterhalb der Oberfläche liegt.

Ein Konstanter Bias kann dabei helfen die Rundungsfehler (Grund: 1) zu verhindern. Jedoch tritt weiterhin Selbstverschattung an Oberflächen auf die fast parallel zur Lichtrichtung liegen. An diesen Oberflächen kann Selbstverschattung nur durch einen sehr großen Bias-Wert verhindert werden. Dies führt jedoch zu einem Artefakt, dass Peter Panning heißt (Abbildung: 3.6). Dadurch das wir die Tiefenwerte der Shadow Map in Richtung des Lichtes mit einem Bias-Wert verschieben, verschieben wir auch die Schatten selbst. Dieses Artefakt ist sehr unangenehm, den wie in Abschnitt 1 erläutert, geben uns Schatten Informaitonen über Skalierung und Position, wenn wir also Schatten verschieben dann führt das zu einer irreführenden Darstellung der Szene.

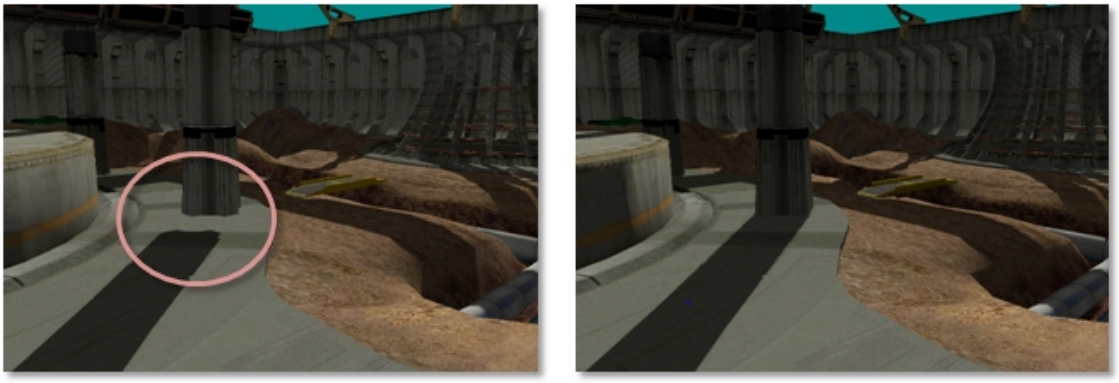

Abbildung 3.6: Links : Szene mit Peter Panning Artefakt. Es führt oft dazu das es so scheint, als würden Objekte schweben.
Rechts: Szene ohne Peter Panning Artefakt.

3.4.1 Heuristisches Biasing

Wie wir im letzten Kapitel gesehen haben kommt es ohne einen bias zu Selbestverschattung und falls ein zu großer Bias verwendet wird kommt es zu Peter-Panning. Es sei außerdem zu bemerken, das bei Oberflächen die orthogonal zu der Lichtrichtung liegen ein nur sehr geringer Bias benötigt wird, wobei für Oberflächen die parallel zu der Lichtrichtung liegen ein großer bias nötig ist. Diese Beobachtung lässt sich ausnutzen indem der Bias Wert proportional zu dem sinus des Winkels zwischen der Lichtrichtung und der Oberflächen Normalen gewählt wird. Anstatt, den Winkel direkt auszurechnen, lässt sich dies effektiv mit dem Standard-Skalarprodukt berechnen. Ein solcher Bias wird Slope-Scale-Bias genannt. Eine alternative Möglichkeit den proportionalen Bias einzusetzen heißt Normal Offset Shadow Mapping. Dabei wird die Oberfläche anstatt, in die Richtung des Lichtes, in die Richtung des Normalen Vektors verschoben.
Beide dieser Verfahren verwenden die selbe Heuristik wobei die Stärke des Bias proportional zu $1 - \overrightarrow{l} * \overrightarrow{n}$, wobei $\overrightarrow{l}$ die Lichtrichtung und $\overrightarrow{n}$ der Normalenvektor der Oberfläche ist. Es sollte erwähnt werden, dass für Oberflächen die orthogonal zu der Lichtrichtung liegen der heuristische Bias 0 ist, deshalb werden diese Biasing Methoden oft mit einem konstanten Bias kombiniert.

Das große Problem mit heuristischen Biasing Methoden ist, dass sie für jede Szene neu konfiguriert werden müssen, denn falls der Bias zu groß ist, kommt es zu Peter-Panning und falls er zu klein gewählt wird kann es zu Selbestverschattung kommen. Um die Konfiguration leichter zu machen werden oft Kombinationen von unterschiedlichen Biasing

Methoden verwendet wie zum Beispiel :
Slope-Scale-Bias + Normal-Offset-Shadow-Mapping + Konstanter Bias.

3.4.2 Alternative Techniken

Eine Alternative zu einem Bias Wert heißt Second Depth Shadow Mapping. Dieses funktioniert jedoch nur für Scenen mit solider Geometrie. Dabei wird in der Shadow Map anstatt der Geometrie die am nähesten zum Licht ist, die Geometrie abgespeichert die am zweit nähesten zum Licht ist. Dies lässt sich effektiv implementieren, indem anstatt von Back Face Culling, Front Face Culling verwendet wird. Dadurch kommt es zu keiner Selbestverschattung mehr, da die Oberfläche die am nähsten an der Lichtquelle liegt immer im Licht ist, denn beim Vergleich der Tiefenwerte werden keine zwei Werte verglichen, die nah beieinander liegen. Es könnte jedoch Selbestverschattung auf der Rückseite der Geometrie auftreten. Da die Geometrie solide ist, kann für Oberflächen die von dem Licht wegzeigen direkt angenommen werden das sie im Schatten liegen. Außerdem tritt kein Peter-Panning auf, da kein Bias verwendet wird. Doch wie bereits erwähnt erfordert Second Depth Shadow Mapping, das die Geometrie der Szene solide ist. Doch auch mit solider Geometrie kann Selbestverschattung auftreten. Wenn die Geometrie sehr dünn ist dann, dann ist der Tiefenunterschied zwischen Front und Back Face zu gering, wodurch beim Vergleich wieder Selbstverschattung auftritt.

3.5 Schattenkanten Antialiasing

In der Implementierung des Shadow Mapping Algorithmus aus Abschnitt 3.1 wird für jeden Oberflächenpunkt binär klassifiziert, ob der Punkt im Schatten liegt. Dies führt jedoch bei geringer Shadow Map Auflösung zu Aliasing. Eine Möglichkeit diesen Effekt zu verringern ist Schattenkanten zu glätten. Dabei ist zu beachten, dass bekannte Antialiasing Algorithmen wie MSAA oder SSAA nicht einsetzbar sind. Wenn auf die Shadow Map direkt ein solcher Filter angewandt wird, so Filtern wir die Tiefenwerte und nicht den Anteil des Lichtes der einen Punkt von einem Licht erreicht.

Percentage Closer Filtering(PCF) [RSC87] ist ein einfacher Algorithmus der Schattenkanten glättet. Dabei wird bei PCF nicht die Shadow Map selbst gefiltert, sondern die Klassifikation, ob ein Pixel im Schatten liegt. Der Algorithmus wird für jeden Oberflächenpunkt ausgeführt und lässt sich in 3 Schritte unterteilen:

1. Wie zuvor wird die Position des Punktes im Light Clip Space Koordinatensystems bestimmt.
2. Der Vergleich der Tiefenwerte wird nun jedoch für alle Pixel der Shadow Map in einem $N \times N$ Feld durchgeführt.
3. Der Durchschnitt der binären Vergleiche stellt den Anteil des Lichtes dar, das den Punkt erreicht.

Der Parameter N beschreibt dabei die Auflösung der Schattenglättung.

Dadurch das bei PCF für jeden Punkt N^2 Shadow Map Pixel gelesen werden müssen gehört PCF zu den langsameren Antialiasing Algorithmen. In [RSC87] wird dieses Problem bemerkt und es werden stochastische Sampling Methoden vorstellt. Dabei werden die Pixel, die für den Vergleich verwendet werden, stochastisch mittels Monte Carlo Sampling gewählt.

Es ist jedoch zu bemerken, dass selbst mit Monte Carlo Sampling, PCF kein effektiver Algorithmus ist und in dieser Ausarbeitung nur aus Gründen der Vollständigkeit enthalten ist. Die Schattenglättung in der Echtzeitberechnung ist ein großes Thema mit vielen

Algorithmen, dazu gehört Variance Shadow Mapping, Exponential Shadow Mapping, Logarithmic Shadow Mapping, Percentage Closer Soft Shadows und noch einige mehr auf die ich hier nicht eingehen werde.

3.6 Bestimmen eines engen Licht Frustums

Die Auflösung der Shadow Map steht im direkten Verhältnis zur Größe des Frustum. Die Tiefe des Frustums (Abstand zwischen Near und Far Plane) ist relativ zu der Tiefenauflösung der Shadow Map. Der Bereich, der von einem Pixel der Shadow Map beschrieben wird, ist relativ zu der Breite und Höhe des Frustum. Ist das Frustum groß, so beschreibt ein Pixel der Shadow Map einen großen Bereich in der Welt.

Daraus folgt, dass eine Möglichkeit die Qualität der Schatten zu verbessern das bestimmen eines möglichst kleinem Licht Frustum ist. Dabei sollte für das Licht Frustum gelten:

1. Alle Oberflächenpunkte die Schatten werfen können sind in dem Licht Frustum enthalten.
2. Die Form des Licht Frustum bleibt gleich.
3. Die Größe des Licht Frustum ist minimal.

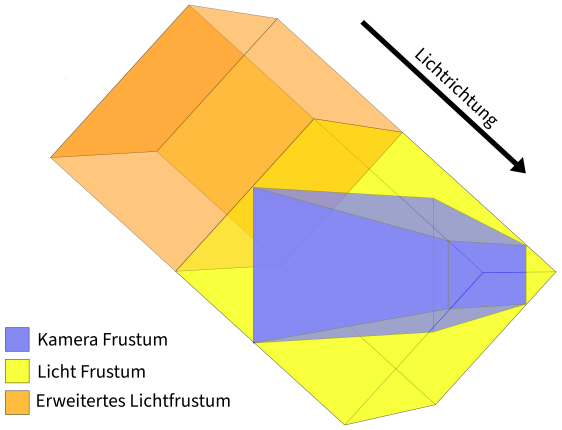

Abbildung 3.7:

Eine einfache Implementierung hiervon funktioniert wie folgt:

1. Transformiere alle Punkte des 2x2x2 Würfels im Clip Space mit der Inversen Kamera Projektion in View Space. Die Eckpunkte des Würfels werden dabei auf die Eckpunkte des Kamera Frustum abgebildet.
2. Transformiere das Kamera Frustum in Light View Space.
3. Bestimme die maximale und minimale Z-Komponente des Kamera Frustum in Light View Space.
4. Die Far Plane ist nun durch die maximale Z-Komponente definiert und die Near Plane durch die minimale. Dadurch das Punkte die außerhalb des Kamera Frustums liegen ebenfalls Schatten werfen können, wird die Near Plane oft nach hinten verschoben.

Hier wird sichergestellt, dass alle Punkte die im Kamera Frustum enthalten sind, ebenfalls im Licht Frustum enthalten sind. Dies ist unnötig, den nicht alle Punkte im Kamera Frustum müssen auch Schatten werfen. Ein minimaleres Frustum kann bestimmt werden, indem zuerst eine Bounding Box um alle Objekte berechnet wird, um anschließend das Frustum so zu wählen, dass der Schnitt zwischen Objekt Bounding Boxes und dem Kamera Frustums vollständig im Licht Frustum enthalten ist.

Bei Lichtern mit einer Perspektivischer Projektion, kann nur die Near und Far Plane angepasst werden, da sich sonst die Form der Projektion ändert. Dadurch kommt nur zu einer Verbesserung der Tiefenauflösung. Für Lichter mit Orthogonaler Projektion kann ebenfalls, die effektive Pixel Auflösung der Shadow Map verbessert werden. Für solche Lichter kann das Kamera Frustum, wie bei Lichtern mit Perspektivischer Projektion, in Light View Space transformiert werden, um anschließend eine Bounding Box um das Kamera Frustum zu bestimmen. Für große Direktionale Lichter, wie die Sonne ist dies besonders wichtig, da sonst die gesamte Szene in der Shadow Map enthalten wäre, wodurch die Auflösung der Shadow Map extrem gering wäre.

3.7 Perspective Shadow Mapping

Eine Möglichkeit um Perspektivisches Aliasing zu verhindern ist das verzerren des Shadow Map Koordinatensystems. Dabei wird das Koordinatensystem so verzerrt, dass Oberflächen die nahe an der Kamera liegen in dem Lichtkoordinatensystem größer sind und dadurch eine höhere Auflösung haben. Eine mögliche Implementierung davon heißt Perspective Shadow Mapping (PSM) [SD02]. Hier wird das Lichtkoordinatensystem mit der Kameratransformation selbst transformiert, wodurch die Perspektive der Kamera ebenfalls auf das Lichtkoordinatensystem angewendet wird. Dadurch ist das Perspektive Aliasing uniform auf alle Pixel verteilt, wodurch der Einfluss auf die visuelle Qualität minimiert wird. Dieses Verfahren kommt jedoch mit einigen Problemen:

- Die Schattenberechnung wird nun in Post Perspective Space durchgeführt (auch Normalized Device Space genannt). Hier müssen Direktionale Lichter als Punkt Lichter und Punkt Lichter als Direktionale Lichter interpretiert werden. Das führt dazu, dass das bildliche Denken in diesem Raum sehr schwierig ist.
- Es tritt weiterhin Projektives Aliasing.

Eine Verbesserung von PSM heißt Light Space Perspective Shadow Mapping (LiSPSM) [WSP04]. Hier wird anstatt der Kameratransformation der Raum mit einer einfacheren Verzerrungsmatrix transformiert. Dadurch kann ebenfalls Perspektivisches Aliasing verhindert werden und durch die einfachere Wahl der Matrix muss die Schattenberechnung zwar immernoch im Post Perspective Space durchgeführt werden, aber jetzt bleiben Direktionale Lichter, Direktionale Lichter. Die Stärke der Verzerrung hängt nun von einem Parameter n ab. In dem gleichen Paper in dem LiSPSM vorgestellt wurde, wird ebenfalls eine optimale Wahl für n berechnet die Projektives Aliasing minimiert : $n_{opt} = near + \sqrt{near * far}$. LiSPSM entfernt demnach Perspektivisches Aliasing und auch Projektives Aliasing kann durch LiSPSM in vielen Fällen minimiert werden.

3.8 Cascaded Shadow Mapping

TODO find original PAPER that introduced CSM.
Cascaded Shadow Mapping (CSM) ist ein weiteres Verfahren um Perspektives Aliasing zu vermeiden. Im Vergleich zu LiSPSM wird der Raum bei CSM nicht verzerrt, sondern in N Räume partitioniert (Abbildung X.Y).

Der Algorithmus lässt sich in folge Schritte unterteilen:

1. Partitioniere das View Frustum in N Räume. Teile dafür das View Frustum an N Oberflächen die orthogonal zu der Near und Far Plane liegen. Eine Möglichkeit diese Flächen zu verteilen ist exponentiel : $z_i = N * (far/near)^{i/N}$
2. Berechne eine enge Projektion [3.6] um jede Partition des View Frustums. Zeichne anschließend die Szene mit den N berechneten Projektionen. Die N erzeugeten Shadow Maps heißen Kaskaden.
3. Zeichne die Szene aus der Kamera Perspektive. Die Schatten berechnung muss dabei leicht angepasst werden. Bei dem Vergleich, im Shadow Mapping Algorithmus, muss jetzt der Tiefenwert mit allen Kaskaden verglichen werden.

Es ist zu bemerken, dass CSM eine Diskretisierung von LiSPSM, denn falls $N \to \infty$ sind beide Verfahren identisch. Durch die Diskretisierung kommt es dazu das der Übergang zwischen Kaskaden oft sichtbar ist. Es ist Möglich LiSPSM zu verwenden um die einzelnen Kaskaden zu zeichnen, doch dies führt dazu das die Schattenberechnung wieder in Post Perspektive Space durchgeführt wird. Eine andere Möglichkeit ohne die Form des Lichtfrustums zu verändern ist eine andere Verteilung der Kaskaden
$z_i = \lambda N(far/near)^{i/N} + (1-\lambda)(n + (i/near)(far - near)$
Dabei ist λ ein Faktor, der die Stärke der Korrektion vorgibt.

Der Große Vorteil von CSM ist, dass die berechnung der Schatten nicht in Post Perspective Space wie bei PSM oder LiSPSM durchgeführt wird. Dadurch können Punkt Lichter als Punkt Lichter und Direktionale Lichter als Direktionale Lichter behandelt werden. CSM eignet sich außerdem für große Lichtquellen wie beispielsweise die Sonne, die die gesamte Szene beleuchten. Solche Lichter benötigen eine hohe Auflösung und da die Auflösung in CSM auf N Texturen aufgeteilt wird können hierfür beispielsweise 4x 4000x4000 Texturen verwendet werden. Diese Auflösung ist mit LiSPSM nicht möglich da viele Grafikkarten keine 16000x16000 Texturen unterstützen. Es sollte jedoch bei CSM nicht vergessen werden, das es Perspektivisches Aliasing nicht so gut entfernt, wie LiSPSM, da CSM eine Diskretisierung von LiSPSM ist. In der Praxis ist die Diskretisierung von CSM jedoch selten erkennbar, weshalb CSM verbreiteter wie LiSPSM ist [SWP11].

4. Fazit

Das schreibe ich am Ende. Aktuell würde ich ehrlich gesagt noch gerne auf VSH eingehen, weil PCF ja nicht wirklich Echtzeitfähig ist und ich würde gerne eine Echtzeitfähige Kombination von Algorithmen vorstellen.

Literaturverzeichnis

[RSC87] William T. Reeves, David H. Salesin und Robert L. Cook: *Rendering Antialiased Shadows with Depth Maps.* In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, Seite 283–291, New York, NY, USA, 1987. Association for Computing Machinery, ISBN 0897912276. https://doi.org/10.1145/37401.37435.

[SD02] Marc Stamminger und George Drettakis: *Perspective Shadow Maps.* In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, Seite 557–562, New York, NY, USA, 2002. Association for Computing Machinery, ISBN 1581135211. https://doi.org/10.1145/566570.566616.

[SWP11] Daniel Scherzer, Michael Wimmer und Werner Purgathofer: *A Survey of Real-Time Hard Shadow Mapping Methods.* Computer Graphics Forum, 2011, ISSN 1467-8659.

[Wil78] Lance Williams: *Casting Curved Shadows on Curved Surfaces.* In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, Seite 270–274, New York, NY, USA, 1978. Association for Computing Machinery, ISBN 9781450379083. https://doi.org/10.1145/800248.807402.

[WSP04] Michael Wimmer, Daniel Scherzer und Werner Purgathofer: *Light Space Perspective Shadow Maps.* In: Alexander Keller und Henrik Wann Jensen (Herausgeber): *Eurographics Workshop on Rendering.* The Eurographics Association, 2004, ISBN 3-905673-12-6.