

# Matrjoschka

Michal Domanski 0347138 932 [sulik@milchzentrale.org](mailto:sulik@milchzentrale.org)

Michael Hanzl 0525742 932 [michael.hanzl@student.tuwien.ac.at](mailto:michael.hanzl@student.tuwien.ac.at)

## Effekte:

- Refraction Shader

Bei unserem Refraction Shader handelt es sich um einen recht simplen Fake. Hierbei wurde keine Referenzimplementation verwendet. Um die Performance zu verbessern setzen wir für unsere Szene Deferred Shading ein, welches uns hier etwas entgegengekommen ist.

Die Glaskugel wird hierbei in einem separaten Pass einzeln gerendert. In den Framebuffer werden die Normalen der Kugel im Viewspace gespeichert. Der Depthbuffer, welcher mit der Deferred Shading Stage geteilt wird, wird mit den Kugeltiefendaten beschrieben. Der Framebuffer wird per Iteration auf den Vectorwert ( 1, 1, 1 ) gelöscht.

Im finalen Renderpass werden nun einfach die Texture Lookups zu den Daten der Deferred Stage einfach um einen Offset verzerrt, der von der entsprechenden Normalen an dem aktuellen Pixel abhängt. Die Offsetfunktion ist hierbei ein reziproker Wert der z Komponente der ausgelesenen Normalen, welche die ersten beiden Komponenten der Normalen gewichtet :

```
float refractOffset = 1.0f / refractNormal.z - 1.0f;  
vSrc.tCoord += refractOffset * refractNormal.xy / 30.0f;
```

Der resultierende Offset Vektor wird aus kosmetischen Gründen um eine Konstante skaliert. Diese Vereinfachung führt in unserem Szenario zu akzeptablen Ergebnissen, da wir hauptsächlich an der Verzerrung der Szene innerhalb der Kugel interessiert sind. Hier halten sich die Tiefenwerte in einem kleinen Intervall, weshalb man ihren Einfluss auf den Offsetvektor ignorieren kann.

- Cascaded Shadow Mapping [1]

Dieser Schattenalgorithmus wurde wie im Paper vorgeschlagen implementiert. Allerdings wurden dort einige Dinge unter den Teppich gekehrt. Die Near Plane der Orthogonalprojektion ist im Optimalfall von dem der Lichtquelle nächstentfernten Objekt abhängig, während im Paper selbige aus der Bounding Box des Kamera Frustum resultiert. Hierbei werden aber alle Shadow Caster, welche sich außerhalb des Kamera Frustum befinden, nicht mitgerendert. Erstaunlich ist, dass bei der entsprechenden Skizze zu dieser Situation alles richtig dargestellt wurde.

In unserem Fall war die Situation insofern erschwert, da wir aus diversen Gründen keine Bounding Boxen für unsere Objekte auslesen. Aus diesem Grund wird die Nearplane auf einen Minimalwert gesetzt, auf Kosten von Tiefenpräzision sowie letztendlich auch Performance ( da kein Culling aller Objekte außerhalb des Lichtfrustum angewandt wird ).

Dennoch führt der Algorithmus zu schönen Ergebnissen.

- SSAO Hack [2]

Dieser wohl bekannte Algorithmus wurde nach dem Ansatz von Crytek

Wir verwenden den Tiefenbuffer aus der Deferred Stage, um hieraus die Szene zu rekonstruieren. Anschließend werden im Viewspace zufällige Positionen innerhalb eines vorgegebenen Radius ermittelt. Hierzu verwenden wir, wie im Paper vorgeschlagen, eine Oberflächennormale, welche wir aus einer dreikomponentigen Random Textur sampeln. Anschließend verwenden wir eine vorgegebene Konfiguration von Vektoren mit einer Länge  $\leq$  dem Kugelradius, und reflektieren diese an der ermittelten Oberfläche. So soll eine geringe Korrelation der Vektorkonfigurationen unter benachbarten Pixeln erreicht werden.

Die Viewspaceposition wird um die so ermittelten Vektoren verschoben, und in den Screenspace projiziert. Nun werden die Tiefenwerte miteinander verglichen. Liegt der projizierte Punkt hinter der gesampelten Tiefe, so befindet man sich vermutlich innerhalb eines Objekts. Analoges gilt für den umgekehrten Fall.

Unter der Annahme, dass die Samplepositionen innerhalb des vorgegebenen Radius gleichverteilt sind, erhält man so einen ungefähren lokalen Verdeckungskoeffizienten innerhalb einer Kugel, welcher den ambienten Teil des an dem Mittelpunkt auftreffenden Lichts abdämpft. Der Einfluss einer Sampleposition auf die Verdeckung wird mit dem reziproken Abstand zur aktuellen Viewspace Position gewichtet.

Wie erwartet führt dieser Approach zu vielen Artefakten. Zum einen führen Objekte, die einen großen Tiefenabstand zu dahinterliegenden Objekten haben, zur Abdunkelung selbiger. Dem kann man mit einem Threshold entgegenwirken, allerdings führt dies in manchen Situation zu einer zu geringen Anzahl an gültigen Samples, weshalb das Ergebnis dennoch nicht zufriedenstellend ist.

Weiters leidet dieser Ansatz an Rauschen. Die Stärke des Rauschens hängt natürlich von der Anzahl der Samples ab, in unserem Fall wurde das Resultat aber erst ab 64 Samples angenehm, was in der Praxis zu unperformant ist. Wir haben versucht, dem mit einem Gauss Filter entgegenzuwirken, allerdings hat sich das Resultat selbst bei einer Filtergröße von 7x7 nur minimal verbessert.

Das Ergebnis war nicht zufriedenstellend. Für bessere Resultate sollten die Szenennormalen berücksichtigt werden, da man nur Interesse an der Verdeckung innerhalb der Hemisphere an der Oberfläche hat. Weiters können so Kanten erkannt werden, welche für die Glättung des Rauschens hilfreich sein können.

#### Quellen:

[1] Rouslan Dimitrov: Cascaded Shadow Maps, August 2007, ["http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded\\_shadow\\_maps/doc/cascaded\\_shadow\\_maps.pdf"](http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf)

[2] Martin Mittring: Finding Next Gen – CryEngine 2, In: Advanced Real-Time Rendering in 3D Graphics and Games Course, SIGGRAPH 2007, Chapter 8, <http://delivery.acm.org/10.1145/1290000/1281671/p97-mittring.pdf?key1=1281671&key2=9942678811&coll=ACM&dl=ACM&CFID=15151515&CFTOKEN=6184618>

[84618](#)