

„Chroan Productions Demo“

Abgabe 2

Christoph Weiler, 1029175, 066 937, christoph.weiler@tuwien.ac.at
Oana-Aurora Moraru, 1108261, 066 932, tsunade_oana@yahoo.com

Treatment

Chroan Productions Demo veranschaulicht eine Szene, die in einem edlen Billiardraum stattfindet. Die Kamerafahrt zeigt zuerst die Szene und zeigt dann die Effekte genauer.

Effekte und Details

- *Shadow Mapping mit Percentage-Closer Filtering Korrektur* - alle Objekte aus dem Raum sollen Schatten werfen
- *Normal Mapping* - für die Säulen
- *Reflections/Environment Mapping* - zwei Hasenstatuen auf jeweils einem Podest. Eine reflektiert die Szene, die andere bricht das hindurchfallende Licht.

Implementierungsdetails

Shadow Mapping mit Percentage-Closer Filtering Korrektur:

Das von uns implementierte Shadow Mapping verfügt über 2 Passes. Das heißt, dass unsere Szene aus 2 verschiedenen Perspektiven gerendert wird: ein mal aus der Perspektive der Lichtquelle und ein zweites mal aus der View/Camera Perspective.

Im ersten Pass wird die gesamte Szene aus der Sicht der “Sonne” gerendert, wobei aber nur die Depth Werte in einer Textur gespeichert werden. Diese Textur wird in einem Framebuffer gespeichert und dann dem Shader übergeben.

Im zweiten Pass wird die Szene aus der Perspektive der Camera gerendert. Beim Zeichnen der Pixel durch den Shader wird jedoch überprüft ob die Entfernung von der Lichtquelle zum entsprechenden Shadow Map Wert gleich der Entfernung von der Lichtquelle zum eigentlichen Pixel ist. Falls das der Fall ist, liegt der Pixel in der Sonne und muss nicht schattiert werden. Falls aber die Distanz zur Shadow Map kürzer als die Distanz zum Pixel selbst ist, liegt der Pixel im Schatten und muss dunkler gezeichnet werden.

Durch das PCF erhält man einen schöneren, nicht so abrupten Übergang vom Schatten in die beleuchteten Flächen. Dies setzt die Verwendung einer `sampler2DShadow` statt einer normalen Textur (`sampler2D`) im Shader voraus.

Normal Mapping

Unser Normal Mapping täuscht komplexe Geometrie durch eine Normal Map vor. Die Säulen in unserem Demo wirken dadurch viel detailreicher als sie in Wirklichkeit sind. Dazu werden die Lichtberechnungen im Tangent-Space durchgeführt.

Zuerst wird der Tangent-Space erstellt. Dazu verwenden wir die Normale, die Tangente und eine Bitangente jedes Vertexes, um in den Tangent-Space des jeweiligen Vertex wechseln zu können.

Der Fragment-Shader liest Normalen der vorgetäuschten Oberfläche aus der Normal-Map aus (Diese Normalen befinden sich schon im Tangent-Space) und wir berechnen ein Blinn-Phong Shading mithilfe dieses Vektors (Anstatt der Normalen, die sich aus dem Modell ergeben).

Die High-Poly Säule haben wir dabei heruntergeladen. Das Low-Poly Modell wurde von uns mit Blender erzeugt und die NormalMap mit Blender selbst gebacken.

Environment Mapping

Das von uns implementierte Environment Mapping verfügt über 2 Passes. Das heißt, dass unsere Szene aus 2 verschiedenen Perspektiven gerendert wird: einmal aus der Perspektive des reflektierenden Objektes, oder genauer gesagt aus dem Mittelpunkt dieses Objektes, und ein zweites Mal aus der View/Camera Perspective.

Im ersten Pass wird die gesamte Szene aus der Sicht des Mittelpunkts des reflektierenden Objektes gerendert. Das Resultat wird in einer cubemap Textur gespeichert, die an einen Framebuffer abgehängt wird und dann dem Shader übergeben.

Im zweiten Pass wird die Szene ganz gewöhnlich aus der Perspektive der Kamera gerendert, jedoch findet zusätzlich noch eine Berechnung des Einfallsvektors vom Augpunkt zur Oberfläche statt.

Reflection:

Für den Reflection Effekt wird der Einfallsvektor in Weltkoordinaten an der Normalen reflektiert und somit erhält man den Reflektionsvektor. Dieser wird dann für den Lookup aus der Cubemap Textur angewandt und man erhält somit die Reflexion des entsprechenden Fragments. Nun kann man eventuell noch zwischen eigentlicher Textur des Objektes und berechnetem Reflexionswert mittels eines reflectivity Faktors interpoliert werden. Als Endresultat erhält man in diesem Fall ein Objekt, das so aussieht, als würde es aus einem Chrom-ähnlichen Material bestehen.

Refraction:

Für den Refraction Effekt wird der Einfallsvektor in Weltkoordinaten an der Normalen gebrochen und somit erhält man den Vektor, der die Richtung des

gebrochenen Lichtstrahls darstellt. Zur Berechnung der Brechung wird ein Ratio verwendet, welcher den Refraction Index des Mediums des Einfallsvektors durch den Refraction Index des Mediums des Brechungsvektors darstellt. Der resultierende Brechungsvektor wird dann für den Lookup aus der Cubemap Textur angewandt und man erhält somit die Nachahmung der Brechnug des Lichts am entsprechenden Fragment. Da wir als Brechungsindex für das zweite Medium 1.5 (Index von Glas) verwendet haben, erhält man als Endresultat ein Glas-ähnliches Objekt.

Kamera

Unser Projekt besitzt 2 Kameraimplementierungen. Die erste ist eine vollautomatische Kamera, die sich selbstständig durch den Raum bewegt.

Die zweite Kamera ist eine First-Person Kamera, die man mit den Tasten asdw und Maus steuern kann. Um zwischen den Kameras zu wechseln, muss die Taste 'c' gedrückt werden.

Die erste Kamera ist eine Catmull-Rom Spline Kamera. Es wurden Keyframes definiert und zwischen ihnen wurde mithilfe einer Catmull-Rom Spline interpoliert. (Position mit Catmull-Rom, View-Point wurde linear interpoliert)

Modellierung

Für die Modellierung, als auch für die Erzeugung der Normal maps haben wir Blender verwendet.

Testing

Wir haben unsere Lösung auf der Nvidia Grafikkarte im Labor getestet.

Quellen

Shadow Mapping:

Vorlesungsfolien CG & RTR

<http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/>

Normal Mapping:

Vorlesungsfolien RTR

<http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/>

http://www.bencloward.com/tutorials_normal_maps1.shtml

http://www.chrisalbeluhn.com/Normal_Map_Tutorial.html

<http://ogldev.atspace.co.uk/www/tutorial26/tutorial26.html>

http://www.poopinmymouth.com/tutorial/normal_workflow.htm

Reflections and Refractions:

Vorlesungsfolien RTR

http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter07.html

http://www.mbroecker.com/project_dynamic_cubemaps.html