Programmierübung 4: Shader Programmierung
Computergrafik, Herbst 2012

Abgabedatum: Donnerstag 15. November, 12:00

In diesem Projekt geht es darum, Ihre Rendering Engine um Funktionalität zum Shading und zur Texturierung zu erweitern. Sie werden dabei lernen, mit GLSL Shader Programmen zu arbeiten. Im Gegensatz zum letzten Projekt, wo Sie Ihren eigenen Rasterer geschrieben haben, kehren wir jetzt zu OpenGL zurück. Erstellen Sie für die einzelnen Teilaufgaben separate Eclipse Projekte, damit Sie diese getrennt demonstrieren können. Sie können jeweils das Projekt "simple" kopieren und dieses dann erweitern.

Diese Übung muss bis Donnerstag 15. November um 12:00 Uhr via Ilias abgegeben werden. Zusätzlich muss die Übung wie üblich einem Assistenten am Computer gezeigt werden. Schreiben Sie sich dazu wieder auf der Online Liste ein.

Bevor Sie anfangen

Wir stellen einen weiteren GLSL Sshader und den dazu passenden Javacode als Beispiel zur Verfügung, um Ihnen den Anfang leichter zu machen. Das Material steht auf Ilias zum Herunterladen bereit. Um den Shader anzuwenden, benutzen Sie die neue Version der Klasse `simple`, welche eine Szene mit einem Würfel mit Texturkoordinaten und Normalen enthält. Weiter ersetzen Sie den Konstruktor von `GLRenderContext` durch den neuen Konstruktor aus dem Zusatzmaterial. Kopieren Sie die Shaderfiles `diffuse.vert` und `diffuse.frag` in Ihr `shaders` Verzeichnis, und die mitgelieferte Textur in ein `textures` Verzeichnis. Nun sollte die neue Version von `simple` lauffähig sein, und Sie sollten einen texturierten Würfel sehen.

Studieren Sie die Shaderfiles und den Javacode, der die nötigen `uniform` Parameter an den Shader übergibt. Eine kurze Erklärung zu den speziellen Variablentypen in den Shaders:

- Typ `uniform`: Grundsätzlich werden `uniform` Variablen im Vertex oder Fragment Shader vom Java Code aus mittels der Funktion `glUniform` übergeben. Alle Variablen die nicht von Vertex zu Vertex oder von Pixel zu Pixel ändern werden so übergeben. Typischerweise gehören dazu die Transformationsmatrizen, Informationen zu Lichtquellen oder Texturen. Der Aufruf von `glUniform` muss erfolgen, nachdem der Shader aktiviert wurde, aber bevor die Vertex Daten via `glDrawArrays` (siehe unten) gezeichnet werden.

- Typ `in` im Vertex Shader: Die `in` Variablen im Vertex Shader werden von Java via sogenannte "Vertex Buffer Objects" an den Shader übergeben. Der passende Java Code ist bereits in `GLRenderContext.draw` implementiert. Das Rendern der Daten wird durch Aufruf der Funktion `glDrawArrays` ausgelöst. "Vertex Buffer Objects" enthalten alle Daten zu den einzelnen Dreieckseckpunkten wie z.B. Positionen, Normalen oder Texturkoordinaten.
- Typ `in` im Fragment Shader, bzw. Typ `out` im Vertex Shader: Die `in` Variablen im Fragment Shader entsprechen den `out` Variablen im Vertex Shader. Diese Variablen werden von OpenGL vom Vertex an den Fragment Shader weitergeleitet und dabei automatisch an jeden Pixel interpoliert.
- Typ `out` im Fragment Shader: Diese Variablen werden in den Frame Buffer geschrieben. Typischerweise ist dies die Farbe des Pixels. Der Shader muss die Tiefenwerte müssen nicht explizit in den Frame Buffer schreiben, diese werden automatisch aufdatiert.

Als Kurzübersicht über GLSL eignet sich die GLSL Quick Reference Card. Bei Fragen bitte vom Ilias Forum gebrauch machen!

1 Lichtquellen und Materialeigenschaften (4 Punkte)

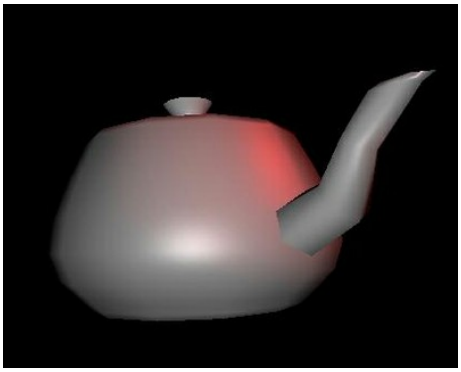
Zuerst soll die `jrtr` Rendering Engine so erweitert werden, dass Lichtquellen und Materialeigenschaften einfach verwaltet werden können. Eigenschaften von Lichtquellen wie in der Vorlesung besprochen werden in der Klasse `Light` gespeichert. Erweitern Sie den `SceneManager`, so dass mehrere Lichtquellen hinzugefügt werden können. Am besten speichern Sie die Lichtquellen in einer Implementation des Java `Collection` Interface. Erweitern Sie Ihre `Shape` Klasse, so dass jedes Objekt ein eigenes Material haben kann. Das heisst, jedes 3D Objekt sollte eine Referenz auf ein Material haben. Die `Material` Klasse soll die Materialeigenschaften speichern und weiter eine Referenz auf einen `Shader` enthalten, der zum Rendern des Materials verwendet wird.

Entwickeln Sie einen Shader, der diffuse Beleuchtung mit mehreren Punktlichtquellen berechnet. Die Parameter der Lichtquellen werden im Shader in `uniform` Variablen gespeichert. Verwenden Sie Arrays, um die Daten mehrerer Lichtquellen aufzunehmen. Die Länge der Arrays kann fix sein, so dass die maximale Anzahl Lichtquellen begrenzt ist. Diese Parameter müssen dem Shader vom Host-Programm mittels der passenden Variante der Funktion `glUniform*` übergeben werden, ähnlich wie im Beispielcode.

Demonstrieren Sie Ihren Shader, indem Sie eine Szene mit mehreren Objekten mit unterschiedlichen Materialeigenschaften (d.h. mit verschiedenen diffusen Reflexionskoeffizienten) und mehreren Lichtquellen konstruieren.

2 Per-Pixel Phong Shading (2 Punkte)

Entwickeln Sie ein Vertex- und Fragment-Shader Programm, das zusätzlich spekulare Reflexion mit per-Pixel Phong Shading mit mehreren Punktlichtquellen ermöglicht. Dieser Shader soll die Beiträge der diffusen und spekularen Reflexion einfach aufaddieren.



Teapot Modell mit spekularer Reflexion von zwei Lichtquellen mit unterschiedlicher Farbe.

Demonstrieren Sie Ihren Shader, indem Sie eine Szene mit zwei Lichtquellen mit unterschiedlichen Farben konstruieren, wie oben im Bild gezeigt. Die zwei Lichtquellen sind am besten unterscheidbar, wenn Sie stark glänzende Materialeigenschaften verwenden.

3 Texturierung (2 Punkte)

Erweitern Sie Ihren Phong Shader, so dass Texturierung und Beleuchtung im selben Shader unterstützt werden. Dazu verwendet man einfach die Farben der Textur als diffuse und ambiente Reflexionskoeffizienten. Implementieren Sie als Erweiterung eine "gloss map", d.h., verwenden Sie Texturwerte, um den spekularen Reflexionskoeffizienten zu bestimmen. Sie können zum Beispiel einfach die Helligkeit der Textur, d.h., die Summe von rot, grün und blau, als spekularen Koeffizienten verwenden.

Demonstrieren Sie diese Funktionalität anhand einer Szene mit mindestens zwei Objekten (oder Oberflächen) mit verschiedenen Texturen. Sie finden viele interessante Texturen, wenn Sie mit Google nach Bildern von "grass texture", "wood texture" etc. suchen. Zum Testen ist stellen wir im Zusatzmaterial auch ein Teapot Modell mit Texturkoordinaten zur Verfügung.

4 Experimentieren mit weiteren Shaders (2 Punkte)

Das Ziel dieser Teilaufgabe ist es, mit irgendeinem weiteren Shader zu experimentieren. Sie dürfen ausdrücklich Shadercode verwenden, den Sie auf dem Internet finden, oder Sie dürfen einen eigenen Shader Ihrer Wahl implementieren. Mögliche Stichworte für einfache Shader sind "toon shading", "procedural brick shader", "procedural stripe shader", oder (für angefressene) "procedural noise shader". Beachten Sie, dass viele Shader, die Sie auf dem Internet finden, möglicherweise für ältere Versionen von GLSL entwickelt wurden und angepasst werden müssen.

Als Resultat sollten Sie den Shader in einer Szene in Ihrem Java Renderer demonstrieren, und Sie sollten den Code erklären können (falls Sie ihn aus dem Internet kopiert haben).