Diplomarbeit

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

Diplom.de

Bibliografische Information der Deutschen Nationalbibliothek:

Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de/ abrufbar.

ISBN: 9783836628716

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

Diplom.de

Diplomarbeit

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

Diplom.de

Heiko Ihde
Shader mit GLSL
Eine Einführung in die OpenGL Shading Language

ISBN: 978-3-8366-2871-6
Herstellung: Diplomica® Verlag GmbH, Hamburg, 2009
Zugl. Technische Fachhochschule Berlin, Berlin, Deutschland, Diplomarbeit, 2009

Inhalt

1 Einleitung

★ **UNSER ZIEL: Grundlagen von GLSL lernen**

Diese Arbeit ist das Ergebnis von dreimonatiger, intensiver Auseinandersetzung mit dem Thema Shader und der OpenGL Shading Language. Sie befasst sich im Groben mit der Erstellung von Shadern mit GLSL und den dafür nötigen Vorbereitungen. Der Autor hat sich während der Erstellung vom blutigen Anfänger zum begeisterten Shader Programmierer entwickelt.

Die Arbeit ist so gestaltet, dass ein Leser mit Grundkenntnissen der Informatik und Computergrafik das Prinzip von Shadern ohne Probleme erlernt. Sichtbare Ergebnisse sollen zum Ausprobieren und Experimentieren ermutigen. Beginnend mit den notwendigen Vorbereitungen werden die wichtigsten Funktionen der GLSL anhand von Beispielen erklärt.

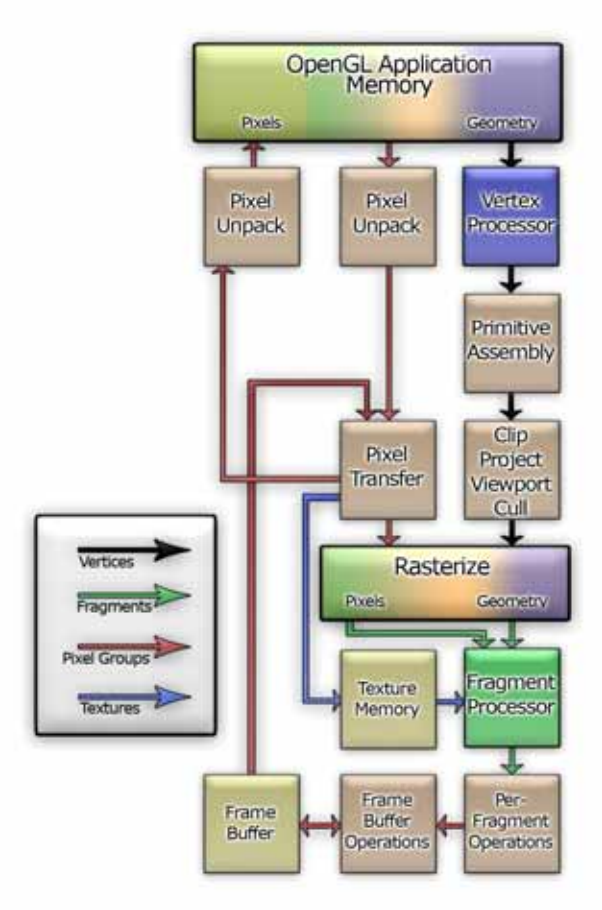
Selbst wer vorher noch nie etwas mit Shadern zu tun hatte, soll nach dem Lesen dieser Arbeit in der Lage sein Shader zu lesen, zu schreiben und diese in OpenGL-Anwendungen zu integrieren.

1.1 Begriffliche Voraussetzungen

Zunächst wollen wir uns mit der Erklärung einiger Begriffe, denen wir im Verlauf dieser Arbeit noch begegnen werden, vertraut machen.

1.1.1 Shader

In der Computer-Grafik wird eine Sammlung von Anweisungen, die anstelle einer festen Funktionalität im GPU (Graphics Processing Unit) auf der Grafik-Hardware läuft, als Shader bezeichnet. Der Shader ist zuständig für die Berechnung der Farbe eines Objektes. Shader wurden bei OpenGL in der Version 1.5 und bei Direct3D in Version 8 eingeführt.



OpenGL Rendering Pipeline
(vgl. [ROS - OpenGL Shading Language] Seite 10)

Dabei wird in der OpenGL Rendering Pipeline die Per-Vertex-Operationen durch Vertex-Shader (läuft auf dem Vertex-Processor), und das Fragment-Processing durch Fragment-Shader (läuft auf dem Fragment-Processor) ersetzt.

(vgl. [WIK2 - Shader])

1.1.2 Shader Objekt

Ein Shader Objekt beinhaltet und verwaltet den Quellcode der einen Shader definiert. Solch ein Objekt kann vom Typ Vertex oder Fragment Shader sein.

(vgl. [ROS - OpenGL Shading Language] Seite 170-172)

1.1.3 Shader Programm

Ein Shader Programm kann aus einem oder mehreren Shader Objekten bestehen. Sie ersetzen je nach Typ die Fixed Functionality komplett. So ersetzt ein Shader Programm, das nur einen Vertex Shader beinhaltet, die Per-Vertex-Operationen. Das Fragment-Processing der Fixed Functionality wird jedoch nicht ersetzt. Andersherum ersetzt ein Shader Programm, das nur Fragment Shader beinhaltet, das Fragment Processing, jedoch nicht die Fixed Functionality der Per-Vertex-Operationen.

Sind sowohl Vertex als auch Fragment Shader im Shader Programm enthalten, so müssen gegebenenfalls die gewünschten Funktionen der festen Funktionalität im Shader per GLSL implementiert werden.

> Im Kapitel **Simulation der Fixed Functionality mit ShaderGen** schauen wir uns ein Programm an, das uns hilft, die Fixed Functionality in einem Shader zu simulieren.

(vgl. [OGL1 - Shading Language Documentation])

1.1.4 Fixed Functionality

Benutzen wir kein Shader Programm, wird ein 3D-Objekt in OpenGL anhand der Fixed Functionality berechnet. Mit dem Setzen von States (Zuständen) kann das Ergebnis dieser Berechnung beeinflußt werden.

Um mehr Einfluss auf Ergebnis und Performance der Berechnung nehmen zu können, ersetzen wir nun diese feste Funktionalität durch eigene Programme. Durch Shader.

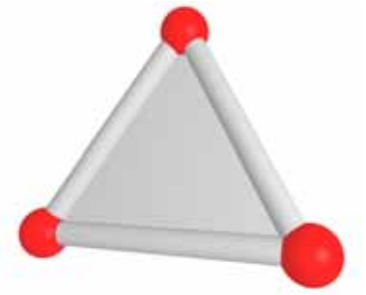
Im Kapitel **Die Shader programmieren** folgen einige Beispiele, in denen wir die Fixed Functionality durch Shader ersetzen werden.

> Mehr Informationen über die Fixed Functionality und OpenGL States können wir in den OpenGL Spezifikationen und Dokumentationen auf **http://opengl.org** bzw. **http://opengl.org/documentation/specs** finden.

(vgl. [OGL1 - Shading Language Documentation])

1.1.5 Vertex

Ein Vertex ist ein Eckpunkt. Aus mehreren Vertices setzt sich das 3D-Modell zusammen. Ein Vertex-Shader wird einmal pro Vertex ausgeführt. Also im Falle dieser Abbildung dreimal. In OpenGL können wir für jeden Vertex eine eigene Normale definieren.



Die Vertices (rot)

1.1.6 Edge

Zwischen den Vertices verlaufen die Edges. Sie beginnen und enden in einem Vertex. Die Edge ist eine Kante eines Faces.

Die Edges (rot)

1.1.7 Face

Die Flächen eines Objektes bestehen meist aus vielen Faces. Eine Normale bestimmt, in welche Richtung das Face zeigt. Dadurch können wir ermitteln, welche Seite die Vorderseite und welche Seite die Rückseite ist.

Das Face (rot)