

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

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.

Dieses Werk sowie alle darin enthaltenen einzelnen Beiträge und Abbildungen sind urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsschutz zugelassen ist, bedarf der vorherigen Zustimmung des Verlanges. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen, Auswertungen durch Datenbanken und für die Einspeicherung und Verarbeitung in elektronische Systeme. Alle Rechte, auch die des auszugsweisen Nachdrucks, der fotomechanischen Wiedergabe (einschließlich Mikrokopie) sowie der Auswertung durch Datenbanken oder ähnliche Einrichtungen, vorbehalten.

Copyright © 2009 Diplom.de
ISBN: 9783836628716

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

Heiko Ihde

Shader mit GLSL

Eine Einführung in die OpenGL Shading Language

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

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtes.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Die Informationen in diesem Werk wurden mit Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden und der Verlag, die Autoren oder Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für evtl. verbliebene fehlerhafte Angaben und deren Folgen.

© Diplomica Verlag GmbH
<http://www.diplomica.de>, Hamburg 2009

Inhalt

1 Einleitung	6
1.1 Begriffliche Voraussetzungen	6
1.1.1 Shader	6
1.1.2 Shader Objekt	7
1.1.3 Shader Programm	8
1.1.4 Fixed Functionality	8
1.1.5 Vertex	9
1.1.6 Edge	9
1.1.7 Face	9
1.1.8 Fragment	10
1.1.9 Vertex Shader	10
1.1.10 Geometry Shader	10
1.1.11 Fragment Shader	10
1.1.12 Object-Space	11
1.1.13 Eye-Space	12
1.1.14 Diffusemap / Colormap	12
1.1.15 Alphamap / Transparencymap	13
1.1.16 Bumpmap	13
1.1.17 Heightmap / Displacementmap	13
1.1.18 Normalmap	13
1.1.19 Specularitymap / Reflectivitymap	13
1.1.20 Luminositymap	14
1.1.21 Environmentmap / Reflectionmap	14
2 Gründe für Shader mit GLSL	15
2.1 Über Shader	15
2.2 Die Shadertypen	15
2.3 Die High Level Shading Languages	16
2.3.1 HLSL	17
2.3.2 Cg	17
2.3.3 GLSL	18
3 Shader in OpenGL	20
3.1 Die OpenGL Shading Language	20
3.1.1 Unterschiede zwischen Vertex und Fragment Shader	20
3.1.2 Unterschiede zu C/++	21
3.1.3 Typecasting und Konstruktoren	21
3.1.4 Neue Datentypen	21

3.1.5 Zugriff auf Vektor-Komponenten	22
3.1.6 Die attribute, uniform und varying Qualifiers	22
3.2 Die OpenGL Shading Language API	23
4 Vier Schritte zum eigenen Shader	25
4.1 Die Objekte vorbereiten	25
4.1.1 Vorbereiten in Blender	25
4.1.2 Vorbereiten in OpenGL	29
4.2 Die Texturen vorbereiten	32
4.2.1 Diffusemap	33
4.2.2 Alphamap	38
4.2.3 Bumpmap	41
4.2.4 Normalmap	43
4.2.5 Huemap	44
4.2.6 Lightnessmap / Brightnessmap	48
4.2.7 Environmentmap	49
4.2.8 Kanten entfernen	54
4.2.9 CrazyBump	55
4.3 Die Shader einbinden	57
4.3.1 Shader erstellen und kompilieren	57
4.3.2 Shader verwenden	59
4.3.3 Werte übergeben	60
4.3.4 Texturen übergeben	61
4.3.5 Das Resultat	63
4.3.6 Die Shader in GLSLDemo einbinden	63
4.4 Die Shader programmieren	66
4.4.1 Farbe	67
4.4.2 Fragmente verwerfen	69
4.4.3 Direktionales Licht	69
4.4.4 Punktlicht	73
4.4.5 Drei Punktlichter	74
4.4.6 Drei Punktlichter (optimiert)	76
4.4.7 Glanz	78
4.4.8 Prozedurale Textur 2D	81
4.4.9 Prozedurale Textur 3D	84
4.4.10 Selektiv Fragmente verwerfen	86
4.4.11 Prozedurale Textur 3D (optimiert)	88
4.4.12 Einbinden von Textur-Maps	92
4.4.13 Nebel	94

4.4.14 Simulation der Fixed Functionality mit ShaderGen	96
4.4.15 Mehrfarb-Lack	97
4.4.16 Hue-Shifter	99
4.4.17 Displacement	104
4.4.18 Displacement Wasser	106
5 Fazit und Aussicht	109
6 Eigenständigkeitserklärung	110
7 Anhang 1: Literaturverzeichnis	111
8 Anhang 2: Verwendete Software	112
8.1 Programmierung	112
8.2 Grafik	112
8.3 Sonstiges	113
9 Anhang 4: GLSL Kurz-Referenz	114
9.1 Data types	114
9.2 Data type qualifiers	114
9.2.1 Global variable declarations	114
9.2.2 Function parameters	115
9.3 Vector components	116
9.4 Preprocessor	116
9.5 Vertex shader variables	116
9.5.1 Special Output Variables	116
9.5.2 Special input variables	117
9.5.3 Varying outputs	117
9.5.4 Attribute inputs	117
9.6 Fragment shader variables	117
9.6.1 Special Output Variables	117
9.6.2 Special input variables	118
9.6.3 Varying inputs	118
9.7 Built-in constants	118
9.8 Built-in uniforms	118
9.9 Built-in functions	121
9.9.1 Angle and Trigonometry Functions	121
9.9.2 Exponential Functions	122
9.9.3 Common Functions	122
9.9.4 Geometric Functions	123
9.9.5 Matrix Functions	123
9.9.6 Vector Relational Functions	124
9.9.7 Texture Lookup Functions	124
9.9.8 Texture Lookup Functions with LOD	129

9.9.9 Fragment Processing Functions	130
9.9.10 Noise Functions	130
10 Stichwortverzeichnis	131

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.

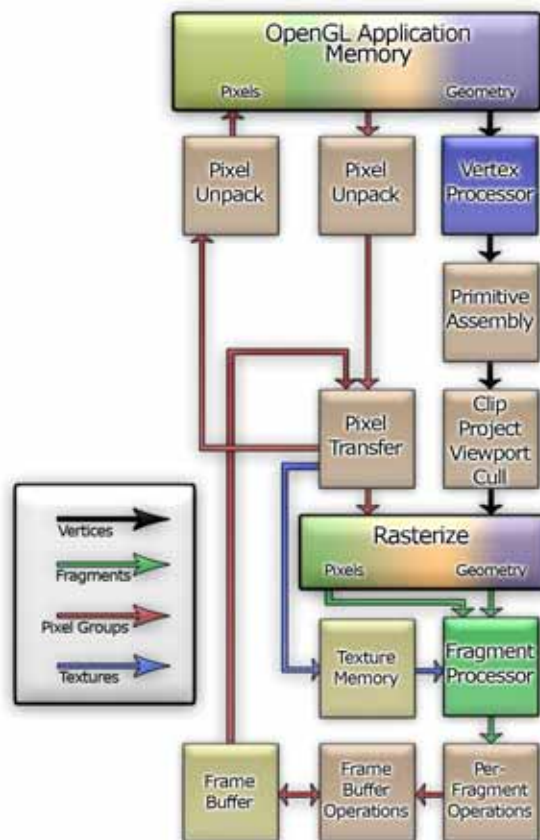
Selbst wer vorher noch nie etwas mit Shadern zu tun hatte, soll nach dem Lesen dieser Arbeit in der Lage sein Shader zu lesen, zu schreiben und diese in OpenGL-Anwendungen zu integrieren.

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 Grafikkarte 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.

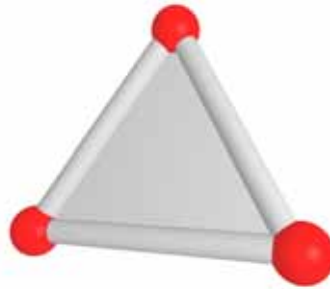
Im Kapitel **Die Shader programmieren** folgen einige Beispiele, in denen wir die Fixed Functionality durch Shader ersetzen werden.

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)