Ein Konzept zur Verbesserung der Darstellungsqualität in der VR-Systemlandschaft der Volkswagen AG

Studienarbeit

Vorgelegt von
Eva Schmidt

UNIVERSITÄT KOBLENZ · LANDAU

Institut für Computervisualistik
Arbeitsgruppe Computergraphik

VOLKSWAGEN AG

IS Konstruktion und Simulation
Design-Systeme und Virtual Reality

Betreuer: Dipl. Inform. Steffen Haupt
Prüfer: Prof. Dr.-Ing. Stefan Müller

April 2004

Vorwort

Innerhalb meines Studienschwerpunktes Computergraphik begann ich mich in den letzten beiden Jahren vermehrt für „Virtual Reality“ zu interessieren und zu faszinieren, nicht zuletzt aufgrund eines Projektpraktikums im Sommer 2003 an der Universität Koblenz-Landau.

So empfand ich es als eine besondere Herausforderung, eine Studienarbeit im Bereich der VR innerhalb eines weltweit agierenden Konzerns, eines „Global Player“, absolvieren zu dürfen, wo VR zur visuellen Überprüfung von Prototypen eingesetzt wird. Die Erschaffung virtueller Welten ist ebenso faszinierend wie anspruchsvoll, denn viele Arbeitsschritte und viel Zeit sind notwendig, um ein Ergebnis zu erzielen, das auch nur annähernd den eigenen Vorstellungen entspricht. Realitätsnähe ist dabei ein entscheidendes Kriterium und wie schwierig es ist, realistisch-wirkende Materialeigenschaften zu simulieren, habe ich bei dieser Arbeit feststellen können.

Viele Eindrücke konnte ich mitnehmen und sehr viel Unterstützung habe ich erfahren. Besonders das herzliche Umfeld und der Respekt, der mir engegengebracht wurde, haben mich tief beeindruckt.

Deshalb gilt mein ganz besonderer Dank all denen, die mich tatkräftig unterstützt und begleitet haben, insbesondere meinen Betreuern bei Volkswagen Dipl. Inform. Steffen Haupt und Dipl. Inform. Frank Purschke und meinem Betreuer an der Universität Koblenz-Landau, Prof. Dr.-Ing. Stefan Müller.

Mein Dank gilt auch denen, die immer ein offenes Ohr für meine Fragen hatten, vornehmlich Andreas Zieringer und Michael Nikelsky, aber auch die Mitarbeiter bei der K-DOE, Jörg Merkl und Oliver Sniehotta.

Zuletzt möchte ich natürlich auch meiner Familie danken, insbesondere meinen Eltern, die mich moralisch und finanziell sehr unterstützten und mir damit die Nutzung dieser Chance erst ermöglichten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der Automobilindustrie verlangt der Markt, bedingt durch die starke Konkurrenz, ständig nach neuen Fahrzeugmodellen. Die Produktzyklen verkürzen sich, für langjährige Entwicklungsprozesse bleibt kaum Zeit.

Um dennoch die Qualität der Produkte zu gewährleisten und dabei Kosten zu senken, werden virtuelle Techniken zu zwei Zeitpunkten im Produktentstehungsprozeß (*PEP*) eingesetzt:

Der erste Meilenstein ist der V1PT, der erste virtuelle Prototyp. Der V1PT ist die digitale Repräsentation des geplanten ersten physischen Prototypen und beschreibt dessen Eigenschaften. Zu diesem Meilenstein gehören das DMU[1] und die ihm zugeordneten Ergebnisse aus Berechnung und Simulation.

Der zweite Meilenstein ist das DDKM, das digitale Datenkontrollmodell. Es stellt alle für den Kunden sichtbaren Elemente eines Fahrzeugs dar und beschreibt die spezifische Einbaulage der Bauteile. Vor allem dient das DDKM auch der Beurteilung der Anmutungsqualität. Mittels einer Präsentation in VR[2] findet hier die Abnahme auf Vorstandsebene statt, aufgrund derer schließlich der erste physische Prototyp[3] gefertigt wird.
Abbildung 1 stellt diese wesentlichen Meilensteine im PEP graphisch dar.

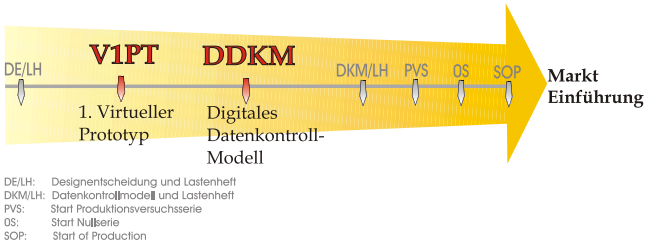


Abbildung 1: Wichtige Meilensteine im PEP sind V1PT und DDKM

Die Funktionalitäten der für solche Präsentationen geeigneten VR-Systeme wurde in den letzten Jahren ständig weiterentwickelt und optimiert, wohingegen die Qualität der visuellen Darstellung bisher zweitrangig war. Um jedoch die Modelle in ihrer realen, emotionalen Anmutung besser beurteilen zu können, muß nun unbedingt der visuellen Darstellung mehr Beachtung geschenkt werden. Die Hoffnung liegt hierbei auch auf dem Einsatz neuer Technologien der Graphik-Hardware, die sich, getrieben durch den Massenmarkt im Consumerbereich, stark entwickelt hat.

[1] digital mock-up
[2] Virtual Reality
[3] DKM-Datenkontrollmodell

1.1 Aufgabenstellung

Das Ziel dieser Studienarbeit ist es deshalb, ein Konzept zur Verbesserung der Darstellungsqualität in VR zu erarbeiten,

- unter Berücksichtigung der Systemlandschaft der Volkswagen AG,
- und der durch neue Generationen an Graphik-Hardware entstandenen Möglichkeiten.

1.2 Einsatz virtueller Techniken bei der Volkswagen AG

Bei der Volkswagen AG finden zu den beiden Zeitpunkten V1PT und DDKM im PEP auf virtueller Ebene Abnahmen statt. Die bei der Konstruktion entstandenen Daten werden dazu entsprechend aufbereitet.
Der folgende Abschnitt stellt die bei der Aufbereitung eingesetzten Systeme vor und geht insbesondere auf die vorhandenen Schnittstellen ein.

1.2.1 Die Prozeßkette

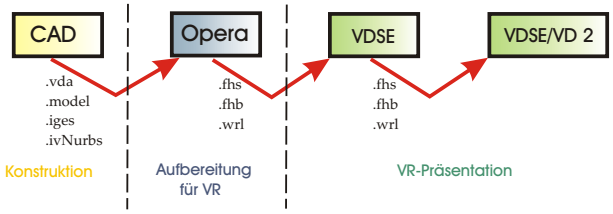


Abbildung 2: Die Prozeßkette bei der Aufbereitung virtueller Fahrzeugmodelle

Konstruktion

Bei der Konstruktion eines Fahrzeugs wird zunächst in CAD-Systemen[4] gearbeitet, in denen auf Basis mathematischer Funktionen Oberflächen modelliert werden. Diese parametrische Darstellung liefert äußerst exakte Beschreibungen der einzelnen Bauteile, die die Grundlage für eine spätere Exaktheit bei der Fräsung bilden.

Modelliert wird dabei mit Hilfe von NURBS[5], rationalen Splinefunktionen mit nicht notwendigerweise äquidistanter Knotenfolge. Sie eignen sich hervorragend zur Modellierung von Kurven- und Freiformflächen.

[4] CAD - Computer Aided Design
[5] Non Uniform Rational B-Splines

Bei Volkswagen werden unter anderem die CAD-Systeme *CATIA* und *IcemSURF* zur Konstruktion eingesetzt.

Aufbereitung für VR

Für eine VR-Präsentation müssen diese parametrischen jedoch in polygonale Daten umgewandelt werden, um eine visuelle Darstellung durch die Graphik-Hardware zu ermöglichen. Dazu müssen die mathematischen Kurven durch eine Dreiecksstruktur approximiert werden (Tesselierung). Je stärker die Krümmung, umso feiner muß ein Dreiecksnetz sein, damit eine Annäherung an die Flächenkomplexität bei möglichst kleinem Sehnenfehler erfolgen kann, wie Abbildung 3 zeigt.

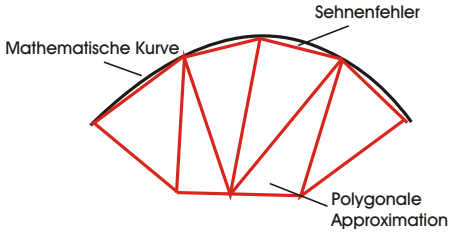


Abbildung 3: Approximation einer Kurve durch ein Polygonnetz

Diese topologiebasierte Tesselierung der Daten findet in dem Datenaufbereitungswerkzeug OPERA statt.
OPERA wurde bei Volkswagen entwickelt und bietet verschiedene Einstellungsmöglichkeiten bezüglich der Tesselierung. Außerdem stehen hier Funktionen für die Texturierung und die Materialvergabe, sowie für die Nachbearbeitung tesselierter Modelle bereit.
Die Optimierung der Daten -bspw. eine Umstrukturierung für ein schnelleres Rendering etc.- erfolgt anschließend durch Xtrans, einem weiteren Werkzeug zur Datenvorbereitung. Xtrans wird von OPERA bei der Konvertierung vom internen Arbeitsformat CSB[6] in das Ausgabeformat aufgerufen.
OPERA importiert alle gängigen parametrischen Datenformate aus CAD-Systemen, wie *vda*, *model* oder *iges*. Die umgewandelten Daten werden anschließend in einem polygonalen Format (*fhs* - Fraunhofer Standard oder *wrl* - VRML2-Format) exportiert.

[6] cosmo binary format

VR-Präsentation

Im Szeneneditor VDSE, der eng mit dem VR-System Virtual Design 2 (VD2) gekoppelt ist (beide Produkte sind von der Firma vrcom), werden die polygonalen Daten geladen und Szeneneinstellungen vorgenommen.
VDSE dient der Vorbereitung einer VR-Präsentation: Unter anderem die Erstellung einer virtuell ausgeleuchteten Szene, in die das Modell integriert wird, sowie die Definition von Kamerapfaden, von Varianten und die Erstellung von Animationen. Über die Benutzeroberfläche von VDSE läßt sich auch die Präsentation mit VD2 steuern.
VD2 ist das eigentliche VR-System, das das Rendering ausführt und die Anbindung an Ein- und Ausgabegeräte übernimmt.

1.3 Offline-Rendering vs. Echtzeit-Darstellung

Sehr viele realistisch anmutende Materialeigenschaften virtueller Modelle sind mit heutigen Algorithmen berechenbar. Beleuchtungsmodelle und Oberflächeneigenschaften realer Gegenstände sind physikalisch so weit erforscht, daß sich die erkannten Gesetzmäßigkeiten mathematisch beschreiben und in Algorithmen umsetzen lassen.
Problematisch sind allerdings die Berechnungszeiten: Je komplexer ein visueller Effekt aufgebaut ist, umso länger dauert meist die Berechnung.

Offline-Rendering

Unter Offline-Rendering versteht man einen Rendering-Vorgang, bei dem nicht die Berechnungsdauer, sondern die visuelle Qualität des Ergebnisses, meist der Photorealismus, im Vordergrund steht. Kaustiken, weiche Schatten, Beleuchtung mittels Radiosity- oder Raytracing-Verfahren, die zum Realismus einer Szene wesentlich beitragen, gehören zu diesen aufwendigeren Algorithmen.
Spezialeffekte in Filmen, aber auch computergenerierte Bilder in Printmedien verlangen diese Qualität. Im Zeitrahmen für solche Produktionen muß also eine entsprechende Zeit für das Rendering veranschlagt werden.

Virtual Reality

Eine Virtual Reality ist eine vom Computer geschaffene, interaktive, dreidimensionale Umwelt, ein immersives „Virtual Environment“.

Immersion bedeutet, daß der Anwender in VR über spezielle Ein- und Ausgabegeräte (Head Mounted Display, Tracker, Stereobrille, Datenhandschuh etc.) annähernd so in die virtuelle Umgebung eingebunden ist, wie in

die „physische“ Realität.

Unter Interaktivität versteht man die Fähigkeit eines Systems, auf Manipulationen durch den Benutzer direkt zu reagieren. Auf diese Weise soll der Benutzer direkt mit den visuellen Objekten in der Virtuellen Realität kommunizieren können.

Abbildung 4: Eine VR-Präsentation an einer Powerwall

Die Anforderung an eine Virtual Reality ist also nicht die einer photorealistischen Abbildung allein, sondern hinzu kommt, daß diese auch in Echtzeit manipulierbar sein bzw. diese sich in Echtzeit verändern können muß. Ab etwa 15 Bildern pro Sekunde (*frames per second - fps*) nimmt der Mensch einen Bewegungsablauf auf einem Bildschirm als flüssig wahr.

Diese Forderung führt jedoch zwangsläufig zu Kompromissen in der Darstellungsqualität, da die meisten Effekte, die physikalisch und visuell überzeugend sind, nicht in der erforderlichen Geschwindigkeit berechnet werden können.

1.4 Aufbau der Arbeit

Eine neue Perspektive, um größeren Realismus auch in VR-Anwendungen zu erzielen, bieten Shadertechnologien der neusten Graphikhardware: Programmierbare Prozessoren in der Graphik-Hardware leisten effiziente Berechnungen von Effekten, die zur Laufzeit auf der Hardware ausgeführt werden.

In dieser Studienarbeit wird deshalb die Einbeziehung von programmierbaren Shadern in Virtuelle Techniken bei Volkswagen und die daraus entstehenden Konsequenzen untersucht.

Zunächst wird in Kapitel 2 auf die zur Verfügung stehenden Shading-Technologien eingegangen und eine Auswahl getroffen.

Kapitel 3 stellt die Testumgebung eines Beispielprojektes, die Algorithmen und die Implementation der ausgesuchten Shader vor.

In Kapitel 4 wird untersucht, inwieweit sich Alias Maya in die Prozeßkette zur VR-Aufbereitung integrieren bzw. zur Erstellung echtzeitfähiger Shader nutzen ließe.

In Kapitel 5 werden Ergebnisse diskutiert und Schlußfolgerungen gezogen.

Kapitel 6 schließt die Arbeit mit einem Ausblick.

2 Shading-Technologien und -Werkzeuge

Dieses Kapitel gibt einen Überblick über die wesentlichen Technologien und Werkzeuge in Bezug auf Shader. Aufgrund einer Gegenüberstellung wird abschließend eine Auswahl getroffen, mit der das Ziel der realistischeren Darstellung in VR-Präsentationen bei Volkswagen weiter verfolgt wird.

2.1 Shading-Technologien

Bis Mitte der 90er Jahre wurde Graphik-Hardware mit speziell angefertigten Chips ausgestattet, die graphische Systeme sehr teuer machten. Mit der Entwicklung von Computerspielen wurde die Massenproduktion von kostengünstiger Graphik-Hardware zur Notwendigkeit.
Die folgenden Abschnitte schildern die Entwicklung der Graphik-Hardware und erläutern deren Programmierbarkeit mit Hilfe von Hochsprachen.

2.1.1 Die Rendering-Pipeline

Beim Rendern einer dreidimensionalen Szene muß jeder 3D-Punkt die Rendering-Pipeline einer Graphikkarte durchlaufen. Hier finden alle Berechnungsschritte statt, die aus einem polygonalen Datenstrom eine Bildschirmausgabe auf Pixelbasis erzeugt.
Ein Polygon wird dabei durch 3D-Punktkoordinaten beschrieben, denen sich Attribute wie Farbwerte, Texturkoordinaten und Punktnormalen zuweisen lassen. Transformationen, Beleuchtungsberechnungen und Texturierungsverfahren müssen zuvor mittels dem Befehlssatz einer API[7], bspw. OpenGL oder Direct3D, definiert werden. Die in Maschinen-Code übersetzten Instruktionen werden dann von der Graphik-Hardware ausgeführt. Abbildung 5 stellt den prinzipiellen Aufbau einer Rendering-Pipeline dar:

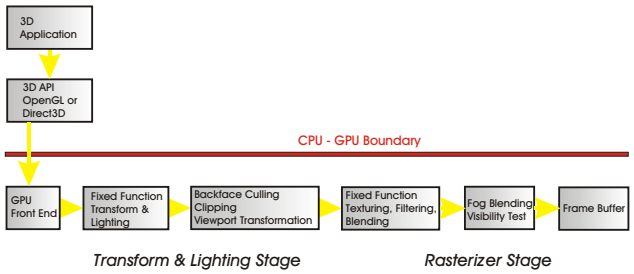


Abbildung 5: Prinzip einer *fixed-function-pipeline*

[7] Application Programming Interface

In der „Transform & Lighting“-Stufe (T & L) finden Transformationen statt, die die Geometrie abhängig von Objekt- und Kameraposition in ein dem gewählten Sichtfeld entsprechendes Koordinatensystem überführen. Geometrie, die außerhalb dieses Sichtfeldes liegt, wird „geclippt“, also nicht weiter mitberechnet. Innerhalb dieses Clipping-Bereiches finden die Beleuchtungsberechnungen statt.

In der „Rasterizer“-Stufe wird die Bildschirmausgabe vorbereitet: Für jedes Polygon der Geometrie werden diejenigen Fragmente bestimmt, die ihm auf einem zweidimensionalen Ausgabegerät entsprechen. Diese Fragmente werden noch texturiert und schattiert, bevor sie schließlich als Pixel auf einem Bildschirm ausgegeben werden können.

Die Graphik-Hardware der beiden letzten Generationen bietet nicht mehr ausschließlich eine feste Rendering-Pipeline (*fixed-function-pipeline*), sondern erweitert diese um zwei Recheneinheiten (Abbildung 6): Den „Vertex-Shader“ und den „Pixel- oder Fragment-Shader“(*programmable-graphics-pipeline*).

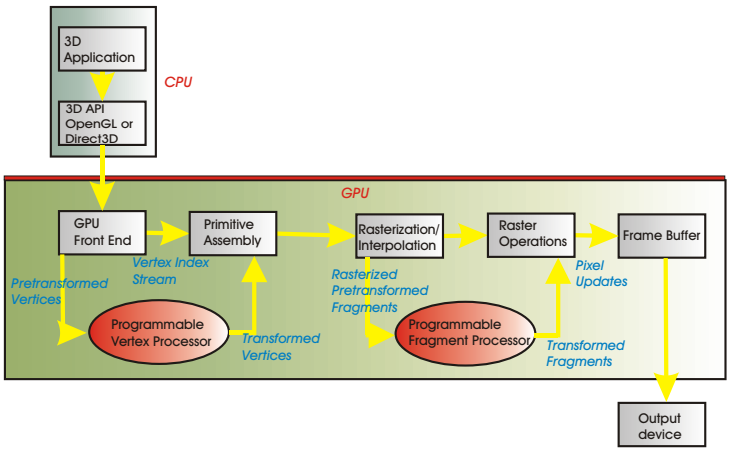


Abbildung 6: Prinzip einer *programmable-graphics-pipeline*

Die beiden Shader-Einheiten machen die Graphik-Hardware zu einer eigenen, äußerst leistungsstarken Prozessoreinheit, der GPU, die mit mehr als 100 Millionen Transistoren doppelt so komplex ist, wie aktuelle PC-Prozessoren.

Der Vertex-Shader ermöglicht es, innerhalb der Rendering-Pipeline, auf Punkt-Ebene Rechenoperationen durchzuführen, der Pixel-Shader ermöglicht Manipulationen auf Pixel-Ebene.

Beide Prozessoren haben eine SIMD-Architektur[8], die es erlaubt, große Datenmengen in kleineren Prozessoreinheiten sehr schnell zu verarbeiten. Sie sind insbesondere für mathematische Berechnungen mit Vektoren ausgelegt (SIMD-Prozessoren werden oft auch Vektorrechner genannt).
Ein Vertex-Programm führt einzelne Berechnungen für jeden Punkt einer Szene aus, beispielsweise Transformationen der Punkt-, Normalen-, oder Textur-Koordinaten.
Ein Pixel- oder Fragment-Programm - ein Fragment wird erst bei der Bildschirmausgabe Pixel genannt, dann verändert es sich nicht mehr - führt Instruktionen für jedes eingelesene Fragment aus. Bei einer Auflösung von 1024x768 Bildpunkten müssen demnach annähernd 800 000 Fragmente pro Frame berechnet werden. Der Aufwand steigt also mit der Auflösung, nicht mit der Anzahl der Vertices.
Viele auf Pixel-Ebene ausgeführte Verfahren liefern im Vergleich zu denen auf Vertex-Ebene die visuell besseren Ergebnisse. In Zukunft wird deshalb die Leistungsfähigkeit der Fragment-Prozessoren sicherlich noch erweitert.

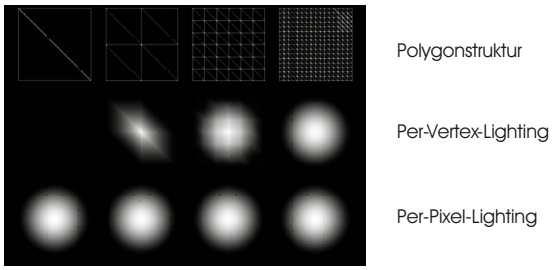


Abbildung 7: Per-Vertex-Beleuchtung ist stark abhängig der Geometrie, Per-Pixel-Beleuchtung nicht

2.1.2 Shading Sprachen

Realistisch anmutende Effekte verlangen jedoch nicht nur verbesserte Hardware-Technologien, sondern vor allem auch deren einfache Bedienbarkeit. Low-Level-Sprachen wie Assembler sind in ihrer Syntax zu schwerfällig, zu wenig intuitiv, da sie zu nah am Maschinen-Code sind.
Sogenannte High-Level-Sprachen sind in ihrer Bedienbarkeit in jedem Fall komfortabler. So nehmen Hochsprachen dem Programmierer beispielsweise Operationen zur Speicherallokierung ab.
Die Entwicklung von „High-Level Shading Languages“, wie man solche Hochsprachen für die Graphikprogrammierung nennt, skizziert Abbildung 8.

[8] single instruction multiple data

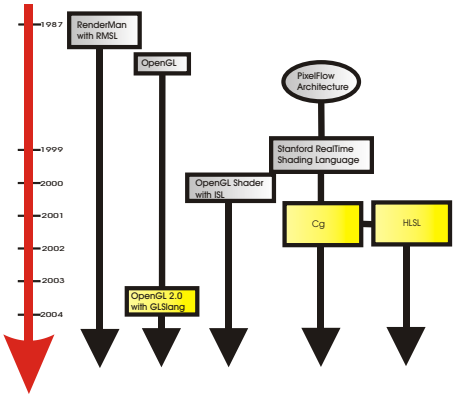


Abbildung 8: Die Entwicklung von High-Level Shading Languages

RenderMan und ISL
Die bei der Firma Pixar entwickelte Shading Language für den Offline Renderer RenderMan ist hier der Vollständigkeit halber aufgeführt; sie erhält nur bedingte Hardwareunterstützung und dient nicht der Programmierung von GPU-Prozessoren, sondern wird in der CPU kompiliert.

Auch ISL[9], die Shading Language der OpenGL Extension „OpenGL Shader“ von SGI, spricht derzeit nicht die Recheneinheiten der Graphik-Hardware an: OpenGL Shader übersetzt in ISL geschriebenen Code in OpenGL-Anweisungen, die mit Hilfe von OpenGL-Erweiterungen in mehreren Rendering-Durchgängen (*Multipass Rendering*) den Shader realisieren. Die so entstandene Struktur wird im Hinblick auf möglichst wenige Rendering-Durchgänge optimiert und findet auf SGI-Hardware beschleunigende Unterstützung. ISL benutzt quasi OpenGL als SIMD-Prozessor, nicht die Shader-Einheiten der GPU.
ISL ist zwar ebenso wie Cg, HLSL und GLSlang eine C-ähnliche Hochsprache, mit der sich Shading-Verfahren umsetzen lassen, jedoch werden ISL-Programme auf der CPU kompiliert. Zudem ist ISL auf die OpenGL-API und das Betriebssystem IRIX festgelegt.

Dem gegenüber bieten in Cg, HLSL oder GLSlang programmierte Shading-Programme folgende Vorteile:

- Kompilierung zur Laufzeit einer Applikation auf den Prozessoren der Graphik-Hardware
- Durch Berechnungen auf Pixel-Ebene verminderte Abhängigkeit von der Anzahl an Geometriepunkten, zunehmende Abhängigkeit von der Auflösung des Darstellungsfensters

[9] interactive shading language

- Kombination von GPU-Prozessoren mit der festen Rendering-Pipeline (Abbildung 6)
- Abhängigkeit nur von der API und der Graphik-Hardware, nicht vom Betriebssystem

Cg und CgFX

Cg[10] ist eine von nVidia entwickelte Shading Language, die an die Programmiersprache C angelehnt ist. Syntaktisch ist Cg allerdings um Vektor- und Matrix-Datentypen, sowie um Texturtypen erweitert. Das Überladen von Funktionen, ein Mechanismus der Programmiersprache C++, wurde ebenfalls integriert. In einer Bibliothek werden bereits einige effizient programmierte Funktionen bereitgestellt, die besonders oft benötigt werden, wie beispielsweise Textur- oder Vektor-Operationen.
Da Cg sowohl für die OpenGL-, als auch für die Direct3D-API kompiliert werden kann, sind Cg-Shader weitgehend portabel. Es bestehen also nur Abhängigkeiten von der API[11] und von der Graphik-Hardware, die über programmierbare Shader-Einheiten verfügen muß.

Die Erkennung der jeweiligen API und der Hardware wird über Profile geregelt: Ein Profil definiert denjenigen Befehlssatz aus der Sprache Cg, der von einer bestimmten Hardware verstanden wird. Eine Übersicht über die derzeit vorhandenen Profile gibt Abbildung 9.

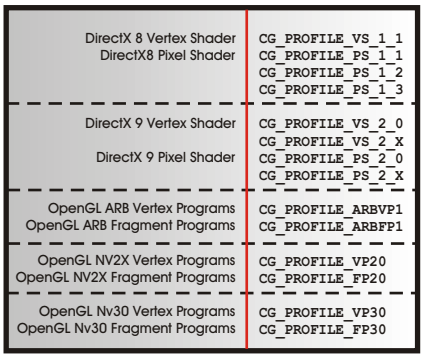


Abbildung 9: Überblick über die Profile in Cg

Deklariert man zum Beispiel in einer Applikation das Profil „CG_PROFILE_ARBVP1“, so wird der für die OpenGL-API bereitstehende Befehlssatz verwendet. Dieser wird von Graphik-Hardware aller

[10] C for graphics
[11] Direct3D oder OpenGL

Firmen verstanden, die sich beim Bau nach dem vom ARB[12] festgelegten OpenGL-Standard richten.

CgFX ist ein Shader-Spezifikations- und Austauschformat, das ebenso wie Cg die OpenGL- und die DirectX-API unterstützt. Das Dateiformat ist identisch mit Microsofts .fx Effekt-Format für DirextX 9, bietet darüber hinaus jedoch den Vorteil der Portierbarkeit auf die OpenGL-API.

Eine CgFX-Datei kann einen kompletten Rendering-Effekt beschreiben: Neben dem eigentlichen Shading-Algorithmus, der in Cg oder in Assembler enthalten sein kann, können auch verschiedene Techniken und Fallback-Varianten implementiert sein, beispielsweise für verschiedene Profile. Ebenso lassen sich in CgFX Effekte umsetzen, die mehrere Render-Passes benötigen. Sehr komfortabel können editierbare Parameter und GUI-Beschreibungen untergebracht werden, die von Digital Content Creation Software (siehe Abschnitt 2.3) interpretiert wird. CgFX ist deshalb auch das geeignete Format für Schnittstellen.

HLSL

HLSL[13] wurde von Microsoft in Koproduktion mit nVidia entwickelt und ist prinzipiell syntaktisch die gleiche Sprache wie Cg. HLSL allerdings ist als Bestandteil von Microsofts DirectX-API zu verstehen und kann auch nur für dessen Graphik-API Direct3D kompiliert werden. Eine Portabilität auf unix-basierte Betriebssysteme ist somit nicht gegeben, da diese den Standard OpenGL verwenden.

GLSlang

GLSlang oder auch „OpenGL 2.0 Shading Language“ wurde bei 3DLabs entwickelt und ist seit Spätsommer 2003 vom ARB als Standard in OpenGL 2.0 festgelegt worden. In der derzeitigen Version 1.5, die 2003 veröffentlicht wurde, ist es bereits integriert. In OpenGL 2.0 soll die Shading Language dann ein fester Bestandteil sein und viele der derzeitig gebräuchlichen Extensions ersetzen, so daß OpenGL in der Version 2.0 zwar einen erweiterten Funktionsumfang besitzen, dabei aber dennoch übersichtlicher werden soll.

GLSlang bietet im wesentlichen denselben Funktionsumfang wie Cg und HLSL und ist ebenfalls an die Programmiersprache C angelehnt. GLSlang bietet darüber hinaus auch Features, die noch von keiner GPU unterstützt werden, wie beispielsweise datenabhängiges Looping.
Allerdings ist die Entwicklung von GLSlang noch nicht so weit fortgeschritten wie bei Cg oder HLSL: Seit Ende 2003 existieren erste Treiber, die jedoch bisher nicht gesamten Funktionsumfang der Sprache unterstützen.

[12] architecture review board
[13] High-Level Shading Language

2.2 Rendering-Systeme

Ein Rendering-System ist ein Bestandteil eines VR-Systems. Es dient dazu, die Elemente einer komplexen Szene zu organisieren und zu rendern.

2.2.1 Szenengraphen

Ein Szenengraph ist ein gerichteter azyklischer Graph, der aus einer Hierarchie von Knoten besteht. Die Knoten können verschiedene Funktionen kapseln. Beispielsweise

- Geometriedaten referenzieren
- Lichtquellen in der Szene definieren
- Transformationsmatrizen bereithalten
- Materialdefinitionen enthalten
- Verschiedene Texturarten und Textur-Spezifikationen enthalten
- u.a.

Knoten können von anderen Knoten referenziert werden, so daß Daten nicht mehrfach im Baum abgelegt sein müssen.

Die Baumstruktur eines Szenengraphen stellt damit eine einfache Möglichkeit zur Organisation und Hierarchisierung einer graphischen Szene dar.
Abbildung 10 stellt das Prinzip eines Szenengraphen dar.

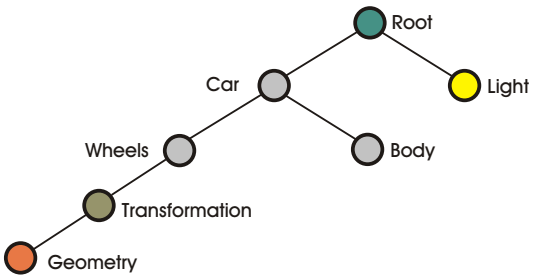


Abbildung 10: Der prinzipielle Aufbau eines Szenengraphen

Beim Rendering wird der Szenengraph traversiert und entsprechend der Knotenfunktion werden die Instruktionen verarbeitet.

Zwei für diese Arbeit relevante, auf Szenengraphen basierende Rendering-Systeme werden in den beiden folgenden Abschnitten kurz vorgestellt:

- Y, als das Rendering-System des bei Volkswagen eingesetzten VR-Systems VD2
- OpenSG, als frei verfügbarer Renderer, dessen Entwicklung von Volkswagen mitfinanziert wurde.

Y

Y ist das Rendering-System in VR-System VD2. Es wurde am IGD (Fraunhofer Institut für Graphische Datenverarbeitung/ Darmstadt) entwickelt.
Y basiert auf OpenGL, der Quellcode ist aber aufgrund der Integration in VD2 nicht frei zugänglich. Derzeit werden Cg-Programme in Y nicht unterstützt.

OpenSG

OpenSG ist ebenfalls ein am IGD in Darmstadt entwickeltes Rendering-System für Echtzeit-Anwendungen [12]. Mit OpenSG Plus wurde das Gesamtprojekt Ende 2003 abgeschlossen [13].

OpenSG ist im Gegensatz zu Y ein offenes System, dessen Quellcode frei zugänglich ist und das nach dem Open-Source Prinzip erweiterbar ist. Es ist so konzipiert, daß es sowohl auf verschiedenen unix-basierten Betriebssystemen, als auch auf Windows genutzt werden kann. Zudem unterstützt OpenSG Cg Shading-Programme.

2.3 Digital Content Creation Software

Insbesondere in der Film- und Computerspielindustrie werden verschiedene 3D-Anwendungen eingesetzt, die zur Modellierung von 3D-Modellen und der Erstellung computergenerierter Szenen, Animationen und Effekte genutzt werden. Doch auch für die Produktion von Still-Renderings[14], die bspw. in Printmedien oder als Werbeplakate veröffentlicht werden, wird DCC[15] Software genutzt.

In diese Kategorie fallen unter anderem die Produkte 3DStudio Max von der Firma Discreet, SoftimageXSI von Avid Technology Inc., Maxons Cinema4D und das Alias-Produkt Maya.

Jedes dieser High-End Programme verfügt über Modellierungswerkzeuge auf parametrischer und polygonaler Basis, über Texturierungs- und

[14] computergenerierter Bilder in meist photorealistischer Qualität
[15] Digital Content Creation

Materialdefinitions-Mechanismen, über umfangreiche Animationsmöglichkeiten und über Renderer, die verschiedene Einstellungsmöglichkeiten zulassen.
Alle ermöglichen das Rendering photorealistischer Effekte mit Radiosity und Raytracing.
Für 3DStudio Max, Maya und SoftimageXSI stellt nVidia jeweils kostenfreie CgFX Plug-Ins bereit.

Bei Volkswagen wird Maya als Digital Content Creation Software vorwiegend im Designbereich eingesetzt.
Deshalb wird Maya im folgenden Abschnitt exemplarisch näher beschrieben.

Maya

Maya ist ein sehr umfangreiches Softwarepaket für hochkarätige Animationen und Still Renderings.

Über die im folgenden beschriebenen Grundfunktionalitäten hinaus bietet Maya noch sehr viele weitere Möglichkeiten und Funktionen. Diese einzeln zu erläutern würde den Rahmen dieser Arbeit sprengen. Deshalb beschränkt sich der folgende Abschnitt auf das Wesentliche.

Aufgrund der sehr komplexen Funktionalität, stellt Maya eine Bedienoberfläche bereit (Abbildung 11):

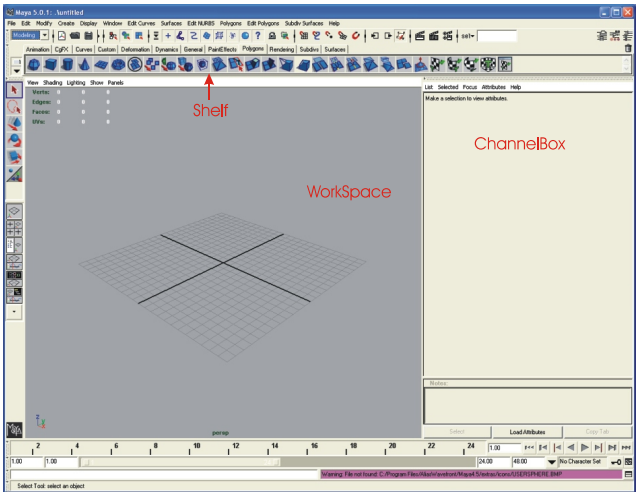


Abbildung 11: Das Benutzer-Interface von Maya

Die Bedienoberfläche teilt sich in einen Arbeitsbereich (*WorkSpace*), eine Werkzeugleiste (*shelf*) und eine Parameter-Spalte (*ChannelBox, AttributeEditor*) auf, über die sich u.a. individuelle Einstellungen bezüglich der

verfügbaren Tools, der Größe, Farbe und Form von Objekten, Lichtquellen, Kameras etc. machen lassen.
Maya ist in 4 Module unterteilt:

1. Modeling
2. Animation
3. Dynamics
4. Rendering

Jedes dieser Module hält spezielle Tools für bestimmte Aufgaben bereit: Tools zur Erstellung NURBS- oder polygonbasierter Modelle, zur Definition von Keyframe- oder Pfadanimationen, zur Erstellung von Partikelsystemen und zur Definition von Beleuchtungsbeziehungen und Schattenwurf.

2.4 Auswahl der Technologien

Die Programmierbarkeit von Shadern in der Rendering-Pipeline stellt eine Innovation im Bereich der Graphik-Hardwaretechnologie dar. Deshalb ist es nicht verwunderlich, daß es viele, teilweise parallele Entwicklungen gibt.

Derzeit ist die Shading Sprache Cg von nVidia die am weitesten entwickelte, bietet sie doch im Gegensatz zu Microsofts HLSL die Unterstützung verschiedener APIs und Plattformen. Somit kann mit Cg auch unter Linux entwickelt werden.

GLSlang ist noch in der Aufbauphase, weshalb sich kein abschließendes Urteil bilden läßt. GLSlang hat jedoch im Vergleich zu Cg die besseren Zukunftsaussichten, da es als zukünftiger Standard in OpenGL 2.0 festgelegt wurde.

Um Cg-Shader auf ihr Verhalten in Echtzeit hin testen zu können, muß eine Testumgebung mit einem Rendering-System erstellt werden. Die Wahl fällt hier auf OpenSG, weil es Cg-Shader unterstützt, als Open-Source System frei zugänglich ist und unter Linux mit einer freien Entwicklungsumgebung implementiert werden kann.

Da Maya bereits bei Volkswagen zur Aufbereitung von Fahrzeugmodellen eingesetzt wird und über eine CgFX-Schnittstelle verfügt, wird zudem untersucht, inwieweit sich Maya auch zur Aufbereitung für die VR nutzen läßt.

Zusammengefaßt lassen sich für das weitere Vorgehen die folgenden Fragestellungen formulieren:

- Wie verhalten sich Cg-Shader in einem VR-System?
- Wie ist die Qualität der Schnittstelle zwischen Maya und Cg?
- Welche Konsequenzen ergeben sich daraus für die Datenvorbereitung?

3 Evaluierung am Beispielprojekt

Um das Verhalten von Shadern in einem VR-System beurteilen zu können, werden in einem Projekt zwei Shader exemplarisch umgesetzt. Als Rendering-System wird eine OpenSG-Umgebung erstellt, in die die jeweiligen Cg-Programme integriert werden.
Implementiert wird unter Redhat Linux Version 8.0 auf einem Intel Xeon PC mit 2,2 GHz Takt, 1 GB RAM und einer nVidia Quadro FX 2000 Graphikkarte.

3.1 Die Testumgebung

Mit Hilfe von OpenSG wird zunächst eine Testumgebung erstellt.
Das Prinzip der Umgebung ist in Abbildung 12 skizziert:

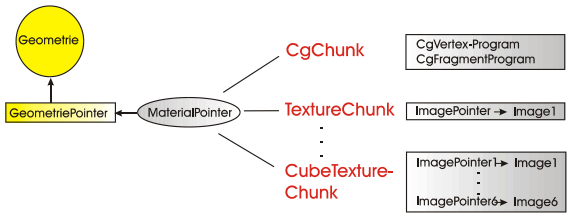


Abbildung 12: Das Rahmenprogramm in OpenSG

Ein Geometrie-Pointer referenziert sowohl die Geometrie (eingelesene VRML-Daten), als auch ein Material. Dieses Material setzt sich aus verschiedenen Attributen, sogenannten Chunks[16], zusammen, wie bspw. Texturen, Cg-Shader etc..

Beim Aufruf werden dem Programm eine oder mehrere Texturen und eine Geometrie-Datei im VRML2-Format übergeben. Die Texturen werden in die TextureChunks eingelesen, die Geometrie wird durch den Pointer referenziert.

Ein Cg-Chunk kann auf alle Textur-Chunks, die zu demselben Material-Chunk gehören, zugreifen, da die Texturen in Registern gehalten werden, die das Fragment-Programm ausliest.

Um das Material der Geometrie zuweisen zu können, müssen bei der Traversierung der eingelesenen VRML-Datei die Geometrieknoten mit dem Material-Pointer verknüpft werden.

Im Rahmenprogramm wird derzeit das ARB-Profil verwendet, so daß im Cg-Vertex-Programm der OpenGL-State ausgelesen werden kann. Dies erleichtert zunächst die Verwendung der notwendigen Transformationsmatrizen und stellt keine wesentliche Einschränkung dar.

[16] Chunk (engl.)= Stück, pl. Einheiten

3.2 Beispiel-Shader

Eine Anwenderbefragung ergab, daß unterschiedliche Problemstellungen in der VR-Aufbereitung existieren. Hierzu gehören beispielsweise die realitätsnahe Darstellung von Aluminium- und Chrom-Flächen, die Visualisierung von Lack, sowie realistischer Schattenwürfe, insbesondere weicher Schattenübergänge.

Derzeit finden VR-Präsentationen auf SGI-Hardware statt, die CLEARCOAT unterstützt. CLEARCOAT ist ein SGI-Produkt, daß der Darstellung von Reflexionen (mittels Environment-Mapping) dient, insbesondere die visuellen Eigenschaften von Lack simuliert. Es ist jedoch nicht auf PC-Systeme protierbar.

Um den vom Anwender gewünschten Szenenrealismus zu erhöhen und nicht auf die IRIX-Plattform beschränkt zu sein, werden in dieser Arbeit Algorithmen für die Darstellung von Aluminium und von Lack untersucht, in Cg implementiert und getestet.

3.2.1 Gebürstetes Metall - Brushed Metal

Modernes und sportliches Design erfordert bei vielen Details den Einsatz von Materialien wie Aluminium oder gebürstetem Edelstahl. Klassisch hierfür sind Aluminiumfelgen, aber auch besonders im Interieur eines Fahrzeuges wird Aluminium und gebürsteter Stahl derzeit häufig verwendet:
In den Armaturen, am Schaltknauf, am Radio, an den Pedalen oder in der Tür.

Abbildung 13: Viele Accessoires im Fahrzeuginterieur bestehen aus gebürstetem Edelstahl oder Aluminium

Sowohl Aluminium, als auch gebürsteter Stahl reflektieren Licht anisotropisch, das bedeutet, daß die Licht-Highlights auf dem Material keine runde, sondern eine langgezogene Form aufweisen. Mathematisch läßt sich dieses Reflexionsverhalten durch eine BRDF[17] ausdrücken.

[17] bidirectional reflectance distribution function - mathematische Beschreibung des Reflexionsverhaltens von Materialien

Physikalischer Aufbau

Eine Aluminium- oder gebürstete Stahloberfläche ist im Gegensatz zu einer polierten Metalloberfläche nicht glatt, sondern weist unzählige, sehr feine Linienstrukturen auf. Diese sogenannten Mikrofacetten streuen das einfallende Licht in viele unterschiedliche Richtungen, so daß sich die Reflexion über einen großen Bereich der Oberfläche erstreckt. Dabei wird teilweise das reflektierte Licht blockiert (Selbstverschattung).

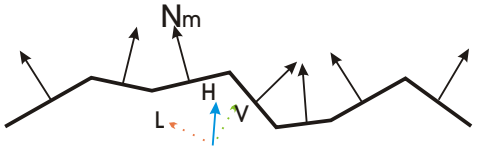
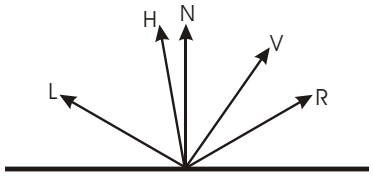


Abbildung 14: Mikrofacetten auf einer Oberfläche. N_m ist die Normale einer Mikrofacette, V und L sind lokaler Sicht- und Licht-Vektor exemplarisch für diese Mikrofacette

Algorithmen

Zur Visualisierung anisotropischer Reflexionen, wie sie u.a. bei gebürstetem Metall, aber auch auf Wasseroberflächen, Haaren oder Fell vorkommen, existieren verschiedene Ansätze. Einige werden im folgenden kurz erläutert. Die Tabelle faßt die in den folgenden Formeln verwendeten mathematischen Symbole zusammen.

N	Normale im Punkt P
T	Tangente im Punkt P
B	Binormale im Punkt P
L	normalisierter, lokaler Licht-Vektor
V	normalisierter, lokaler Sichtvektor
H	Halb-Vektor, berechnet aus L + V
k_d	diffuser Materialparameter
k_s	spekularer Materialparameter



Torrance-Sparrow Modell

Torrance & Sparrow [8] stellten schon 1967 ein theoretisches Modell vor, das annimmt, daß eine Oberfläche aus Mikrofacetten besteht, die als sehr kleine, ebene und perfekte Spiegel fungieren.

Die spekulare Komponente ρ wird dabei durch diejenigen Facetten erzeugt, die in Richtung des Halbvektors H ausgerichtet sind, in diesem Fall

ist die Facetten-Normale dem Halbvektor gleich.
Die von Torrance & Sparrow formulierte Gleichung lautet

$$f_r(L,V) = \frac{k_d}{\pi} + \frac{k_s}{\pi}\rho \tag{1}$$

mit

$$\rho = \frac{FDG}{(N \cdot V)(N \cdot L)}$$

wobei $\frac{k_d}{\pi}$ den diffusen und $\frac{k_s}{\pi}\rho$ den spekularen Term beschreibt. Der spekulare Term besteht aus dem Koeffizienten $\frac{k_s}{\pi}$, einer Fresnel-Komponente F, einem Selbstverschattungsterm G und einem Term D, der die Mikrofacetten-Verteilung auf der Oberfläche beschreibt. Das Teilen durch das Produkt der beiden Punktprodukte sorgt dafür, daß nur diejenigen Mikrooberflächen berücksichtigt werden, die für den Betrachter und das Licht im sichtbaren Bereich liegen.

Der Fresnel-Term F (siehe Abbildung 15) wird durch

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1 + \frac{(c((g+c)-1)^2}{(c((g-c)+1)^2}\right) \tag{2}$$

berechnet, mit

$$c = V \cdot H$$

und

$$g^2 = \eta^2 + c^2 - 1$$

Die Menge des reflektierten Lichtes wird dadurch abhängig vom Betrachtungswinkel, denn bei direkter Draufsicht auf eine Oberfläche ist diese weniger reflektierend, als bei schräger Draufsicht.

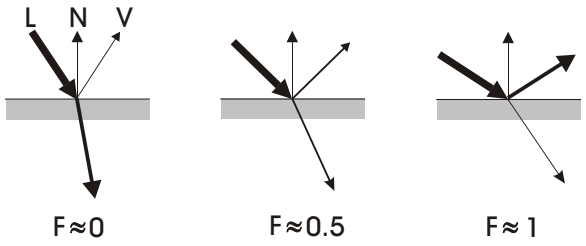


Abbildung 15: Ein Fresnel-Term trägt der Beobachtung Rechnung, daß eine Oberfläche bei schrägem Sichtwinkel stärker reflektiert, als bei steilem Sichtwinkel

G, der Selbstverschattungsterm, wird mit

$$G = \min\left\{1, \underbrace{\frac{2(N\cdot H)}{V\cdot H}(N\cdot V)}_{A}, \underbrace{\frac{2(N\cdot H)}{V\cdot H}(N\cdot L)}_{B}\right\} \tag{3}$$

berechnet. A ist der „masking-term“: Ein Teil des reflektierten Lichtes erreicht den Betrachter nicht; B ist der „shadowing-term “, der den Teil des einfallenden Lichtes beschreibt, der die Oberfläche nicht erreicht.
Die Terme beruhen auf dem V-Groove-Modell (Abbildung 16), das unter den Annahmen gemacht wurde, daß

- sich das Licht symmetrisch, längslaufend und isotropisch-verteilt verhält und
- sich die oberen Kanten der Mikrofacetten in einer Ebene befinden.

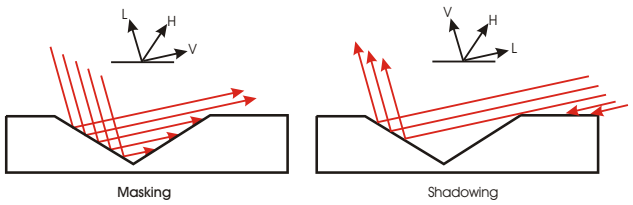


Abbildung 16: Aus dem V-Groove-Modell lassen sich der masking- und der shadowing-term ableiten

Der Mikrofacetten-Term oder „Rauhheitsterm“ D beschreibt die Verteilung der Orientierung von Mikrofacetten, die mit einer Verteilungsfunktion angenähert wird. Beispielsweise

$$D = e^{-\frac{\sqrt{2}}{\beta}\alpha} \tag{4}$$

D wird somit abhängig gemacht von α, dem Winkel zur Oberflächennormale und von β, dem Halbwinkel.

R. L. Cook & K. E. Torrance veröffentlichten 1982 ihre Approximation für den Mikrofacetten-Term [16]. Sie nähern ihn mit der Beckmann‘schen Verteilungsfunktion an

$$D = \frac{1}{4m^2\cos^4\beta}\exp\left(\frac{\tan^2\beta}{m^2}\right) \tag{5}$$

und vereinfachen diese mit

$$\tan^2 \beta = \frac{\sin^2 \beta}{cos^2 \beta} = \frac{1 - \cos^2 \beta}{\cos^2 \beta} = \frac{1 - (N \cdot H)^2}{(N \cdot H)^2}$$

zu

$$D = \frac{1}{4m^2(N \cdot H)^4} \exp\left(\frac{(N \cdot H)^2 - 1}{m^2(N \cdot H)^2}\right) \tag{6}$$

Hierbei ist $m \in [0, 1]$ der Parameter für die Rauheit der Oberfläche.

NDF Shading

Das NDF-Shading Modell, wie es von J.Kautz, K. Daubert und H.-P. Seidel veröffentlicht wurde [6], beruht auf dem Torrance-Sparrow Modell:

$$L_o(L,V) = k_a + \frac{k_d}{\pi} L_i(L)(N \cdot L) + k_s S(L,V)(N \cdot L) \tag{7}$$

mit

$$S(L,V) = L_i(L) \frac{GF\rho(H)}{\pi(N \cdot L)(N \cdot V)}$$

Gegenüber dem Ansatz von Torrance und Sparrow ist ein ambienter Term k_a und ein diffuser Term $\frac{k_d}{\pi} L_i(L)(N \cdot L)$, wie sie bei Phong [15] berechnet werden, hinzugekommen.

Der spekulare Term besteht aus einem spekularen Koeffizienten $\frac{k_s}{\pi}$ und dem Term $S(L,V)$, der den Selbstverschattungsterm G und den Fresnel-Term F beinhaltet, und dazu die Farbe der Lichtquelle $L_i(L)$ multipliziert.

Die Komponente $\rho(H)$ stellt in dieser Gleichung den Mikrofacetten-Term dar: Eine Wahrscheinlichkeits-Dichte-Funktion oder auch Normalen Verteilungsfunktion[18].

Eine NDF ist eine Gauß-Funktion, die abhängig von Erwartungswert und Varianz als Orts- bzw. Skalierungsparameter ist. Anders als in analytischen Modellen wie dem Torrance-Sparrow-Modell wird hier jedoch die NDF nicht berechnet, sondern in eine zweidimensionale Textur projiziert. Eine solche NDF-Textur kann beliebig mit einem Paint-Programm (*Photoshop, Paint Shop Pro o.ä.*) erzeugt werden, so daß der Anwender die Form und Farbe des Highlights selbst bestimmen kann.

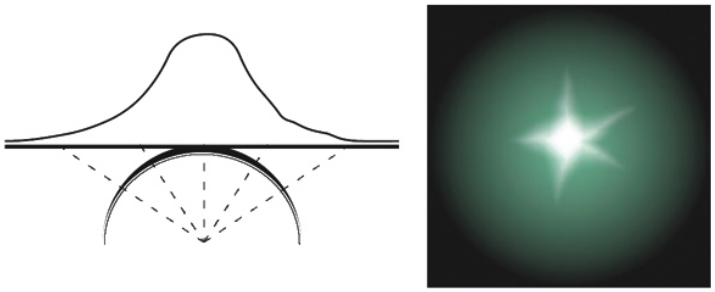


Abbildung 17: *links:* Projektion einer Normalverteilung in eine zweidimensionale Textur, *rechts:* Ein Beispiel für eine NDF-Textur

[18] NDF - Normal Distribution Function

Implementation

Mit dem NDF-Shading-Verfahren wird in dieser Arbeit ein Shader für Aluminium umgesetzt. Es bietet dem Verfahren von Cook-Torrance gegenüber den Vorteil, daß die Mikrofacetten-Verteilung nicht berechnet werden muß. Hieraus ergeben sich im wesentlichen drei Vorteile:

1. Die Berechnung ist schneller, da die aufwendige Berechnung des Rauheitsterms durch ein Textur-Lookup ersetzt wird
2. Es muß kein Parameter m für die Rauheit der Oberfläche angenommen werden
3. Das Verfahren ist nicht auf eine Verteilungsfunktion beschränkt, da die Verteilung beliebig durch die Textur gesteuert werden kann

Für den Fresnel-Term wird die Approximation nach C. Schlick [18] verwendet, die per Vertex berechnet wird.

$$F = f_\lambda + (1 - f_\lambda) * (1 - (H \cdot V)^5) \tag{8}$$

Der Fresnel Koeffizient f_λ ist abhängig von der Wellenlänge des Lichtes und korrespondiert mit der reflektierten Lichtfarbe bei orthogonalem Einfallswinkel.

In dieser Arbeit wird darauf verzichtet, den Koeffizienten für jede Wellenlänge zu berechnen, da dies zu viele Berechnungsschritte nach sich ziehen würde. Stattdessen wird ein konstanter Wert angenommen, der sich aus dem Refraktionsindex n_1 der Lichtquellenumgebung (Luft) und dem des Oberflächenmaterials (Aluminium) n_2 zusammensetzt:

$$f_\lambda = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2}$$

Der Wert des Koeffizienten ist dadurch beinahe 1.

Dieses Vorgehen ist zulässig, da Aluminium bei einer weißen Lichtquelle, ähnlich wie dielektrische Materialien (Plastik, Porzellan), ein annähernd weißes Highlight besitzt. Ein andersfarbiges Metall wechselt die Farbe des Highlights zur Lichtquellenfarbe erst bei flacherem Einfallswinkel.

Selbstverschattungs- und diffuser Term werden nach obigen Formeln ebenfalls im Vertex-Shader implementiert.

Die NDF-Textur wird über die Komponenten des lokalen Halbvektors im Fragment-Shader indiziert und anschließend mit den anderen Komponenten mit dem Faktor 0.5 geblendet.
Durch die Transformation des Halbvektors in das Tangentialkoordinatensystem (siehe unten) durch die entsprechende Matrix pro Vertex, entspricht die X-Koordinate des lokalen Halbvektors dem Punktprodukt mit der

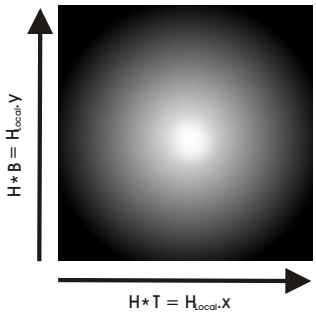


Abbildung 18: Die NDF-Textur wird durch den loaklen Halbvektor indiziert

Tangente, die Y-Koordinate dem mit der Binormale.

Grundvoraussetzung für das Shading ist eine gut aufbereitete Geometrie. Für das Lookup einer NDF-Textur muß ein lokaler Halb-Vektor für jeden Vertex berechnet werden. Um die erforderlichen Vektoren (Licht-, Sicht- und Halb-Vektor) in das Textur- oder Tangentialkoordinatensystem umrechnen zu können, muß an jedem Vertex ein Orthonormalsystem aus Vertex-Normale, -Binormale und -Tangente aufgebaut werden.

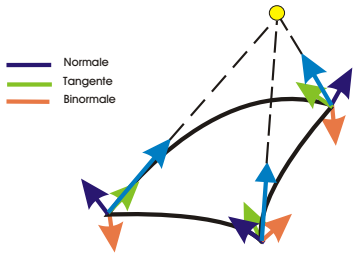


Abbildung 19: Tangente, Binormale und Normale bilden für jeden Vertex ein Texturkoordinatensystem

Aus der Orthonormalität der Vektoren ergibt sich folgende Beziehung:

$$T = Normale \times Binormale$$
$$B = Normale \times Tangente$$
$$N = Binormale \times Tangente$$

Bei parametrischen Daten ergeben sich Tangente und Binormale aus den partiellen Ableitungen nach den Variablen s und t. Für eine Torus-Tangente beispielsweise gilt

$$\frac{\delta_x}{\delta_s} = -2\pi(M + N\cos 2\pi t)\sin 2\pi s$$
$$\frac{\delta_y}{\delta_s} = 2\pi(M + N\cos 2\pi t)\cos 2\pi s$$
$$\frac{\delta_z}{\delta_s} = 0$$
$$T = \left\langle \frac{\delta_x}{\delta_s}, \frac{\delta_y}{\delta_s}, \frac{\delta_z}{\delta_s} \right\rangle$$

mit M als Radius vom Zentrum des Loches zum Zentrum der Torus-Hülle und N als dem Radius der Torus-Hülle.

Bei polygonalen Daten liegen jedoch keine Kurvenparameter s und t vor, Tangente und Binormale müssen anders berechnet werden.

Hier nutzt man die Eigenschaft eines Dreiecks aus, daß alle drei Eckpunkte in einer Ebene liegen. Demnach sind die Tangentialebenen und damit die Tangente und Binormale für alle Punkte gleich.

Verfügen die Eckpunkte über Texturkoordinaten, so kann man einen Vertex als 5D-Vektor beschreiben, der sich aus den Punkt- und den Texturkoordinaten (*UV-Koordinaten*) zusammensetzt:

$$\begin{aligned} vertex_0 &= \langle x_0, y_0, z_0, u_0, v_0\rangle \\ vertex_1 &= \langle x_1, y_1, z_1, u_1, v_1\rangle \\ vertex_2 &= \langle x_2, y_2, z_2, u_2, v_2\rangle \end{aligned}$$

Aus der Planaritätsbedingung des Dreiecks lassen sich die drei folgenden Ebenengleichungen herleiten:

$$\begin{aligned} A_0x + B_0u + C_0v + D_0 &= 0 \\ A_1x + B_1u + C_1v + D_1 &= 0 \\ A_2x + B_2u + C_2v + D_2 &= 0 \end{aligned} \tag{9}$$

Mit Hilfe der 5D-Vektoren können die aufgestellten linearen Gleichungssysteme dann nach den Koeffizienten aufgelöst werden:

$$\begin{aligned} \langle A_0, B_0, C_0\rangle &= (\langle x_0, u_0, v_0\rangle - \langle x_1, u_1, v_1\rangle) \times (\langle x_0, u_0, v_0\rangle - \langle x_2, u_2, v_2\rangle) \\ D_0 &= -\langle A_0, B_0, C_0\rangle\langle x_0, u_0, v_0\rangle \\ \langle A_1, B_1, C_1\rangle &= (\langle y_0, u_0, v_0\rangle - \langle y_1, u_1, v_1\rangle) \times (\langle y_0, u_0, v_0\rangle - \langle y_2, u_2, v_2\rangle) \\ D_1 &= -\langle A_1, B_1, C_1\rangle\langle y_0, u_0, v_0\rangle \\ \langle A_2, B_2, C_2\rangle &= (\langle z_0, u_0, v_0\rangle - \langle z_1, u_1, v_1\rangle) \times (\langle z_0, u_0, v_0\rangle - \langle z_2, u_2, v_2\rangle) \\ D_2 &= -\langle A_2, B_2, C_2\rangle\langle z_0, u_0, v_0\rangle \end{aligned} \tag{10}$$

Tangente und Binormale ergeben sich schließlich aus den Relationen

$$T = \left\langle \frac{\delta_x}{\delta_s}, \frac{\delta_y}{\delta_s}, \frac{\delta_z}{\delta_s} \right\rangle = \left\langle -\frac{B_0}{A_0}, -\frac{B_1}{A_1}, -\frac{B_2}{A_2} \right\rangle$$

$$B = \left\langle \frac{\delta_x}{\delta_t}, \frac{\delta_y}{\delta_t}, \frac{\delta_z}{\delta_t} \right\rangle = \left\langle -\frac{C_0}{A_0}, -\frac{C_1}{A_1}, -\frac{C_2}{A_2} \right\rangle$$

Um die Vektoren in das Texturkoordinatensystem letztlich umrechnen zu können, wird eine Transformationsmatrix aus der normalisierten Normale, Tangente und Binormale erstellt:

$$\begin{pmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{pmatrix}$$

Diese Art der Tangenten-Berechnung setzt Texturkoordinaten voraus.

Da in den zur Verfügung stehenden Geometrie-Daten weder Tangenten, noch Texturkoordinaten für die Vertices vorhanden sind, wird in der Implementierung des NDF-Shading zu einem einfachen Trick gegriffen:

Weil die Kamera und somit der Augpunkt der Szene nicht verändert werden, wird eine globale Y-Achse angenommen, die also für jeden Vertex gleich ist. Die Binormale berechnet sich aus dem Kreuzprodukt zwischen Punktnormale und globaler Y-Achse, die Tangente entsprechend aus Binormale und Normale.

Eine Felge mit NDF-Shading zeigt Abbildung 25.

3.2.2 Metallic-Lack

Ein wichtiges Kriterium im visuellen Erscheinungsbild eines Fahrzeugs ist die Lackoberfläche, sind doch die gesamte Karosserie und oft auch die Stoßfänger vorne und hinten lackiert. Einerseits dient die Lackierung dem Schutz des Bleches vor Umwelteinflüssen, andererseits wird aber auch das spezifische Design der Fahrzeugsilhouette durch die Eigenschaften der Lackschicht modelliert. Insbesondere der Verlauf von Lichtkanten trägt wesentlich zum Erscheinungsbild bei.

Aufgrund dessen muß der Visualisierung eines physikalisch plausiblen und visuell überzeugenden Lackes in einer VR-Präsentation besondere Aufmerksamkeit geschenkt werden.

Physikalischer Aufbau

Ein Lack besteht immer aus mehreren Schichten, die aufeinander folgend aufgetragen werden.

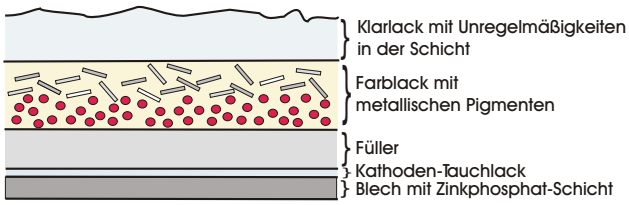


Abbildung 20: Physikalischer Aufbau einer Lackschicht

Zunächst wird das Blech mit einer Zinkphosphatschicht versiegelt, die zusammen mit der anschließenden Grundierung vor allem dem Korrosionsschutz dient.

In einem Grundierungsverfahren legt sich der sogenannte Kathoden-Tauchlack über das Material: In einem Tauchbecken wird hierzu eine Spannung angelegt; bei der Entladung bildet sich ein Teilchen-Film, der auch in Hohlräumen gleichmäßig die Oberfläche bedeckt. Bei großer Hitze wird dieser Teilchenfilm eingebrannt.

Der Füller dient vor allem dem Schutz vor Steinschlag, füllt und glättet aber auch evtl. noch vorhandene Unebenheiten der Metalloberfläche. Die Füllschicht besteht aus Harzen und hat meist eine graue Farbigkeit.

Nun folgt die farbgebende Schicht: Effekt- und Buntpigmente, sowie Bindemittel und andere Zusatzstoffe ergeben einen Basislack. Die Effektpigmente darin sind Aluminium-Flakes oder Glimmerpigmente

(Mica-Pigmente), die mit einer Größe von ca. 5-45 μm bis zu 50 mal größer als die Farbpigmente sind. Diese Effektpigmente fügen einen Reflexionsaspekt zur Materialfarbe hinzu: Je nach Sichtwinkel sieht man mehr metallene Plättchen oder mehr Farbpigmente. Aus der Nähe betrachtet, scheinen Metallic-Lackierungen Tiefen- und Glitzer-Effekte aufzuweisen.

Die abschließende Schicht bildet ein Klarlack. Dieser besteht aus Polymeren und soll Schutz bieten. Die klare Lackschicht muß witterungsbeständig und kratzfest sein, einen UV-Schutz bieten und soll dabei langfristig glatt und hochglänzend bleiben.

Algorithmen

Für den visuellen Eindruck einer Lackschicht sind vor allem die beiden oberen Schichten maßgebend, da die darunter liegenden Schichten meist grau sind und am visuellen Eindruck kaum beteiligt sind.

Multi-Texturing Ansatz

Dumont-Bècle et al. stellten 2001 einen Ansatz vor, der auf Multi-Texturing basiert [14]:

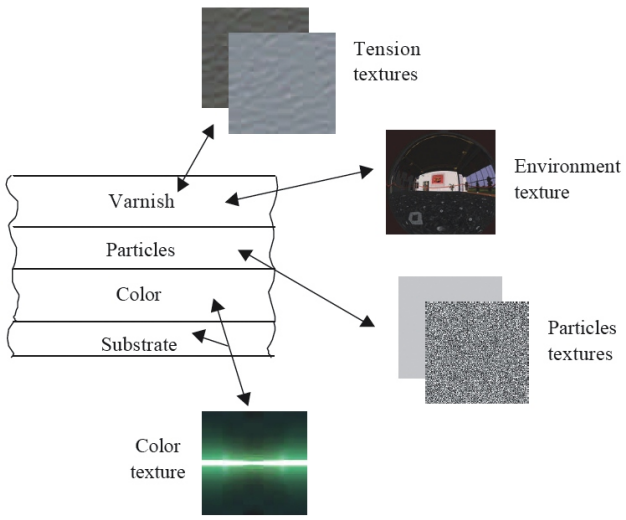


Abbildung 21: Multi-Texturing-Ansatz nach Dumont-Bècle et al.

Die Grundfarbe ist durch eine Textur gegeben, die sich aus goniometrisch gemessenen Farbwerten eines Lackes zusammengesetzt. Sie wird durch die Punktprodukte $L \cdot N$ und $H \cdot V$ indiziert und simuliert dadurch die diffuse Objektfarbe, die abhängig von der Position der Lichtquelle und der des Betrachters ist.

Die Glimmerpigmente werden durch eine Noise-Textur nachgeahmt, die abhängig vom Abstand zum Betrachter stärkeren oder schwächeren Einfluß auf den Effekt hat. Um eine zu starke Abdunklung durch diese Textur zu verhindern, wird sie durch eine weitere Partikel-Textur korrigiert.

Um die Reflexionen der Klarlack-Schicht zu simlieren, wird eine Environment-Textur verwendet. Hierzu wird ein Reflexionsvektor berechnet, der die Environment-Textur indiziert.

Eine letzte Textur (bzw. zwei Texturen im additiven Modus) ist eine Bump-Textur. Sie soll eine feine Relief-Struktur auf der Oberfläche immitieren.

Dieser Ansatz versucht "mit Hilfe mehrerer Texturen, die übereinandergelegt und geblendet werden, die verschiedenen Lack-Schichten zu simulieren. Ein Multipass-Rendering ist hierzu zwingend notwendig, da auf den Ergebnissen der einzelnen Rendering-Passes aufgebaut wird.

Car Paint 9

Im Manual zum CgToolkit [11] veröffentlicht nVidia den Code eines Lack-Shaders, der auf gonioreflektrometrisch vermessenen Lackproben basiert. Die vermessene Lackprobe wird wie im Ansatz von Dumont-Bècle in einer Textur gespeichert, die über die beiden Punktprodukte zwischen Normale und Licht-Vektor bzw. Nomale und Halb-Vektor indiziert wird. Damit bilden die Texturwerte die diffuse, farbgebende Komponente der Lackschicht.

Der spekulare Term wird mit dem Halbvektor des Blinn-Phong-Modelles berechnet [19]:

$$\rho = (N \cdot H)^{shininess} \tag{11}$$

Der Parameter *shininess* liegt wie im Phong-Modell im Intervall $[0, 256]$. Zusätzlich fließt in die spekulare Komponente ein Term ein, der die metallischen Glimmerpigmente simuliert. Hierzu wird eine Normal-Map verwendet, die wie beim Bump-Mapping-Verfahren in der Textur gespeicherte Normalen ausliest und diese zur Berechnung einer spekularen Komponente auf Pixel-Ebene heranzieht. Diese wird abhängig von der Entfernung des Betrachters in die Berechnung mit einbezogen:

$$\begin{aligned} distance &= 1 - \sqrt{VertexPosition^2} \\ distanceAttenuation &= 1 - \exp -distance \end{aligned} \tag{12}$$

Um Unregelmäßigkeiten im Erscheinungsbild des Lackes zu erzeugen, wird ebenfalls eine Normal-Map benutzt. Dies führt zu leichten Störungen des Reflexionsvektors, der in eine Cube-Environment-Map zeigt.

Das Glanzverhalten der Klarlackschicht wird durch Skalierung eines Fresnel-Terms mit der Luminanz der Environment-Map erzielt.

Implementation

Der in dieser Arbeit implementierte Shader lehnt sich im wesentlichen an den CarPaint 9-Shader von nVidia an.

Auf Vertex-Ebene werden zunächst die benötigten Licht-, Sicht- und Halbvektoren berechnet und mit Hilfe der aus Tangente, Binormale und Normale berechneten Transformationsmatrix in das Texturkoordinatensystem überführt (siehe Abschnitt 3.2.1). Dies ermöglicht es, die Beleuchtungsberechnungen auf Pixel-Ebene durchführen zu können.

Der für das Environment-Mapping benötigte Reflexionsvektor wird mit Hilfe der in der Cg-Bibliothek verfügbaren Funktion $reflect(I, N)$ für jeden Vertex berechnet. Die Funktion erwartet als Eingabeparameter den einfallenden Sichtvektor I, der sich aus dem negativen Sichtvektor ergibt, und die Vertexnormale N. Die Funktion $reflect(I, N)$ berechnet den reflexionsvektor R wie folgt:

$$R = I - 2N(N \cdot I) \tag{13}$$

Diese Formel begründet sich durch das physikalische Gesetz, daß der Einfallswinkel des einfallenden Strahls dem Ausfallwinkel des reflektierten Strahls entspricht. Abbildung sowieso verdeutlicht den geometrischen Hintergrund.

Im Vertex-Shader wird außerdem ein Fresnel-Term mit der Approximation anch Schlick [18], wie in Gleichung 8 vorgestellt, implementiert.

Im Fragment-Shader werden die Beleuchtungsberechnungen ausgeführt: Die diffuse Komponente ergibt sich aus einer linearen Interpolation zwischen der diffusen Objektfarbe und dem Produkt der diffusen Objektfarbe mit einem ambienten Term. Das Punktprodukt zwischen Vertex-Normale und Lichtvektor wird hierzu als Blend-Faktor verwendet.

Die diffuse Objektfarbe kann wahlweise die Eckpunktfarbe des Polygons, eine zuvor gesetzte Objektfarbe, die für jeden Eckpunkt gleich ist, oder ein Farbwert aus einer Textur sein, die über $(N \cdot L)$ indiziert wird, sein. Das bedeutet, daß sich goniometrisch vermessene Lack-Werte, die in einer Textur gespeichert wurden, unproblematisch auf das Modell bringen lassen.

Die spekulare Komponente berechnet sich wie im Blinn-Phong-Modell [15], wobei der Exponent über die Größe des Highlights bestimmt.
Die Environment-Map wird mit dem im vertex-Programm berechneten Reflexionsvektor indiziert. Als Environment-textur wird hierzu eine Cube-Map

verwendet (siehe Abbildung 22), die für jede der sechs Seiten eines Würfels[19] eine Textur beinhaltet.

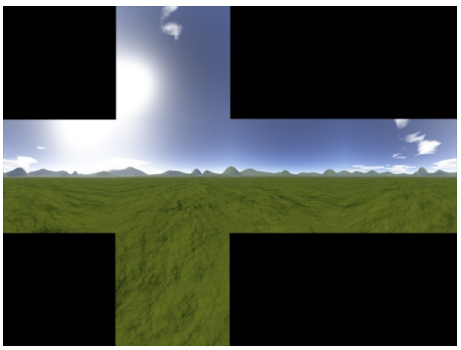


Abbildung 22: Die bei der Implementation verwendete Environment-Textur

Aus einem Punktprodukt eines fest angenommenen Vektors mit dem ausgelesenen Farbwert der Cube-Map ergibt sich ein Fresnel-Wert per Fragment, der mit dem zuvor errechneten eigentlichen Fresnel-Wert multipliziert wird. das bewirkt, daß in den Fresnel-Zonen die Reflexion aus der Environment-Map stärkeren Einfluß auf das visuelle Erscheinungsbild nimmt.

Über diesen Fresnel-Faktor werden diffuse Komponente und Reflexions-Farbwert der Environment-Textur interpoliert und zu der spekularen Komponente addiert.

[19] Cube (engl.) = Würfel

4 Shading mit Maya

Maya hat eine knoten-basierte Programmarchitektur. Es gibt verschiedene Bausteine, die unterschiedliche Attribute haben können, d.h. jedes Element in Maya, sei es eine Kurve, eine Textur, eine Lichtquelle oder ein Material o.a., entspricht einem Knoten. Die Knoten lassen sich beliebig kombinieren, d.h. alle Attribute können miteinander verbunden werden.

Eine solche Verknüpfung einzelner Knoten wird als Dependecy-Graph bezeichnet. Im Hypergraph-Editor kann der Graph überprüft, analysiert oder auch editiert werden.

4.1 Shading Networks in Maya

Ein Shading Network ist ein Ausschnitt aus dem Dependency Graph: Hier werden nur diejenigen Attribut-Verbindungen und Bausteine angezeigt, die für das Shading relevant sind. Hier existieren keine Knoten für die Geometrie oder für Transformationen.

Der Hypershade[20] dient der Erzeugung und Modifikation eines Shading Network.

Materialien bzw. Shading-Verfahren wie Lambert, Phong, Blinn, anisotropisches Shading etc. können über Ein- und Ausgänge beliebig mit 1D-, 2D- und 3D-Texturen verknüpft werden. Texturen wiederum können mit Techniken wie Bump-Mapping oder Displacement-Mapping verbunden werden. Wird ein Shader verwendet, der über mehrere Ebenen gerendert werden soll, so läßt sich der Modus, mit dem diese Ebenen geblendet werden sollen (additiv, subtraktiv, multiplikativ etc.), über sogenannte Utility-Knoten steuern.

Insgesamt existieren unendlich viele Kombinationsmöglichkeiten. Nicht alle sind sinnvoll, doch prinzipiell läßt die knotenbasierte Struktur des Graphen dies zu.

Für jedes Objekt einer Szene können komplexe Shading Networks angelegt werden. Beim Rendering wird der gesamte Dependency-Graph traversiert, ausgelesen und verarbeitet. Je komplexer eine Szene aufgebaut ist und je komplexer die verwendetetn Materialien, umso mehr Zeit wird zum Rendering benötigt. Shading Networks lassen sich demnach nicht in Echtzeit rendern.

4.2 Das CgFX PlugIn in Maya

Mit dem CgFX Plug-In für Maya [10] stellt nVidia eine Schnittstelle zur Verfügung, die Cg-Shader in Maya integrieren.

Das Plug-In besteht aus einem CgFX-Compiler, der die komplette CgFX-Syntax unterstützt, einer CgFX Runtime API, die das Laden und die Mani-

[20] über Window -▷ Hypershade

pulation der CgFX-Files gestattet und dem Modul zur Integration in Maya.

Das hierzu verwendete Austausch- und Spezifikations-Format für die Shader ist CgFX. Der folgende Abschnitt geht auf die Funktionalität des Plug-Ins näher ein.

4.2.1 Funktionen des Plug-Ins

Das Plug-In dient der Visualisierung von Materialien zur Laufzeit eines Programmes durch erweitertes Hardware-Rendering und die Sprache Cg. So muß ein Anwender nicht unzählige Test-Renderings durchführen, um den gewünschten Effekt überprüfen zu können, denn mit Hilfe des Plug-Ins wird ein Effekt direkt im Arbeitsbereich, dem WorkSpace, angezeigt.

Über die Runtime API und das Modul ist das Plug-In derart mit Maya verbunden, daß sich Lichtquellen aus Maya integrieren lassen, im Attribute-Editor verstellbare Parameter für die CgFX-Shader verfügbar sind und Shader ebenfalls in Mayas Hypershade angezeigt werden.

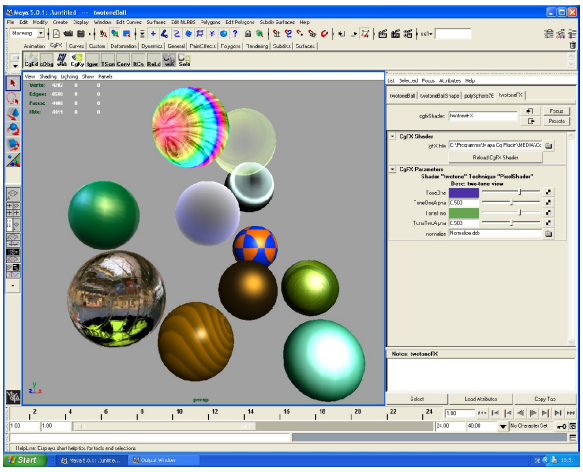


Abbildung 23: Das CgFX Plug-In in Maya: Eine Testszene mit verschiedenen Shadern

Zusammen mit einer Version des Cg-Compiler läßt sich das Plug-In in Maya problemlos nutzen.
Enthalten sind außerdem einige Beispiel-Shader im CgFX-Format: Für Bump-Mapping, Toon-Shading, Anisotropisches Shading, Metall, Dispersions- und Refraktions-Effekte u.a.

Für jeden Shader aus dieser Demo-Bibliothek wird bei Verwendung im Attribute-Editor eine dynamische GUI[21] erstellt. Hier lassen sich (wie bei

[21] graphical user interface

allen anderen Materialien in Maya auch) über Schieberegler (Slider) die shader-spezifischen Attribute verändern.

Diese Parameter werden natürlich über den CgFX-Code und damit die Runtime API gesteuert: In der Datei wird festgelegt, in welchem Intervall der Parameter verstellt, in welchen Stufen eine Einstellung vorgenommen werden kann und welchen Default-Wert er erhält.

4.2.2 Erweiterungen

Eine Erweiterung des Plug-Ins ist durch Mel-Skripte[22] [10] möglich. Die Skripte lassen sich über die Icons im Shelf aufrufen (siehe Abbildung 24).

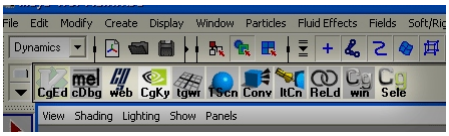


Abbildung 24: MEL-Skripte lassen sich über Icons im Shelf aufrufen

Die meisten dieser MEL-Skripte dienen der Verbesserung und Organisation des Workflows, beispielsweise die Selektierungsfunktion oder die Umschalt-Funktion zur Drahtgitterdarstellung.

Die folgende Tabelle listet die verfügbaren Skripte auf und erläutert kurz deren jeweilige Funktion.

[22] MEL- Maya Embedded Language

MEL-Skripte für das CgFX Plug-In	
CgEd	Öffnet den Code eines ausgewählten CGFX-Shaders in einem Editorfenster
CDbg	Erlaubt es, im Editor bearbeiteten CGFX-Code zu debuggen
Web	Öffnet ein Browserfenster und verbindet den Anwender mit der Website *www.cgshaders.org*, die Informationen zu Shadern enthält
CgKey	Führt alle veränderbaren float-Attribute im selektierten CGFX-Shader in der Channelbox auf
tgwr	Läßt den Benutzer zwischen plastischer und Drahtgitterdarstellung hin- und herschalten
tScn	Erstellt eine Testszene aller in der Shader-Bibliothek zur Verfügung stehenden CGFX-Shader, die in der Szene auf Kugeln dargestellt werden
conv	Übersetzt ein in der Szene angelegtes Shading Network in einen CGFX-Shader, der in laufenden Projektdatei abgelegt wird.
ltcn	Verbindet die Parameter einer Lichtquelle in Maya mit den passenden Slots eines selektierten CgFX-Shaders.
Reld	Lädt einen Shader neu in die Szene, z.B. nachdem er im Editor bearbeitet wurde
Win	Öffnet ein Zusatzfenster, in dem alle in der Szene verwendeten CgFX-Shader aufgelistet werden
Sele	Wählt den Shader des selektierten Objektes in der Szene aus und erlaubt so einen schnellen Zugriff auf die Parameterliste der Channelbox

Die für den Entwickler interessantesten Skripte sind die zur Konvertierung, Editierung und zum Debugging von CgFX-Shadern. Insbesondere die Konvertierungsfunktion würde bei der Datenvorbereitung für die VR eine wichtige Rolle spielen, weshalb sie an dieser Stelle näher untersucht wird.

Konvertierung von Shading Networks

Mit der „Convert“-Funktion lassen sich einfache Shading Networks aus Maya in CgFX-Code übersetzen. Diese werden anschließend in der Projektdatei unter „Shader“ abgelegt[23]. Verwendete 2D-Texturen werden dabei ins DDS-Format[24] konvertiert, sofern sie nicht schon in diesem Format im Projekt vorliegen. DDS ist ein von DirectX unterstütztes, komprimiertes Format. Zum Laden von DDS-Texturen wird eine DirectX-Header-Datei benötigt, folglich ist das Format auf die DirectX-API ausgerichtet.

[23] vorausgesetzt, ein entsprechender Pfad wurde angelegt. Ansonsten wird die Datei im Projektordner gespeichert

[24] Direct Draw Surface

5 Diskussion der Ergebnisse

In diesem Kapitel werden die Ergebnisse aus der Shader-Implementierung und aus der Untersuchung des Plug-Ins zusammengestellt.

5.1 Cg-Shader in OpenSG

5.1.1 Performanz/Echtzeitverhalten

NDF Shading

Das NDF Shading wurde in dieser Arbeit an einer Felge getestet. Hierzu wurde eine originale VRML2-Datei mit zwei bereits für die Visualisierung des gesamten Fahrzeuges positionierten Felgen aus OPERA exportiert. Die Tesselierung wurde bei dieser Testdatei zunächst relativ grob eingestellt.

Die folgende Tabelle listet die Meßwerte für das NDF-Shading bei einer Felge auf, da sich eine einzelne Felge näher heranzoomen ließ. Bei der Darstellung beider Felgen ist die Anzahl der Knoten und Dreiecke natürlich doppelt so hoch, der FPS-Wert ist unwesentlich geringer.
Die Ergebnisse wurden mit der Statistik-Funktion in OpenSG ermittelt.

NDF Shading		
Pixelauflösung	**800x800**	**1280x1024**
Anzahl Knoten	587	587
Traversierungszeit	0.02 Sec	0.02 Sec
Anzahl Dreiecke	12148	12148
FPS	ca. 150 - 200	ca. 85 - 100

Wie sich zeigt liegt die Frame-Rate auch bei bildschirmfüllender Auflösung (und starkem Zoom) immer über 85 FPS; die Darstellung bleibt also auch bei Rotation des Objektes immer flüssig und damit echtzeitfähig.

Lack
Der Shader für Lack wurde am Modell eines VW Polo gemessen, das für eine VR-Präsentation aufbereitet wurde und dementsprechend fein aufgelöst ist. Hierbei wurden jedoch nur diejenigen Karosserieteile berücksichtigt, die beim realen Fahrzeug auch lackiert werden.
Die Tabelle gibt eine Übersicht zu den mit OpenSG gemessenen statistischen Ergebnissen.

Lack		
Pixelauflösung	**800x800**	**1280x1024**
Anzahl Knoten	953	953
Traversierungszeit	0.117 Sec	0.117 Sec
Anzahl Dreiecke	198637	198637
FPS	ca. 7,5	ca. 6,6

Auch wenn bei diesem Modell keine so hohen Frame-Raten erreicht werden wie bei der Felge, so das Ergebnis doch recht zufriedenstellend.

5.1.2 Qualität

Die Qualität der implementierten Shader ist nicht so einfach zu beurteilen. Wie in Abschnitt 1.3 erläutert, sind Echtzeit-Darstellungen im Gegensatz zu Still-Renderings immer mit Kompromissen verbunden. Mit Echtzeit-Shadern die visuelle Qualität von Offline-Shadern zu erreichen, ist relativ schwierig.

Im folgenden werden deshalb die Cg-Shader mit derzeit bei VR-Präsentation zu dem gleichen Zweck eingesetzten Shading-Verfahren verglichen. Die in den Abbildungen 25, 26 und 27 gegenüber gestellten Screenshots wurden in OpenSG mit eingeschaltetem Antialiasing, in VD2 ohne Antialiasing gemacht.

NDF Shading Der visuelle Eindruck des Aluminium-Shaders mit dem NDF-Shading-Verfahren ist überraschend gut. Das Material wirkt matt glänzend, das Highlight ist weiß, vom Licht abgewandte Flächen sind entsprechend abgedunkelt. Erst bei sehr spitzem Sichtwinkel wirkt das Material etwas zu diffus und „grau“.
Da der Shader kein Environment-Mapping implementiert, reagiert er ausschließlich auf die Position der Lichtquelle, die in diesem Fall gleich dem Augpunkt des Betrachters ist.

Im Gegensatz hierzu wirkt die in VDSE mit einem Aluminium-Material belegte Felge zu dunkel. Das für Aluminium typische Grauwert-Spektrum wird nicht ausgeschöpft und trotz eines CLEARCOAT-Shaders auf dem Material sind Reflexionen nur ansatzweise erkennbar.
Abbildung 25 zeigt beide Shader im Vergleich.

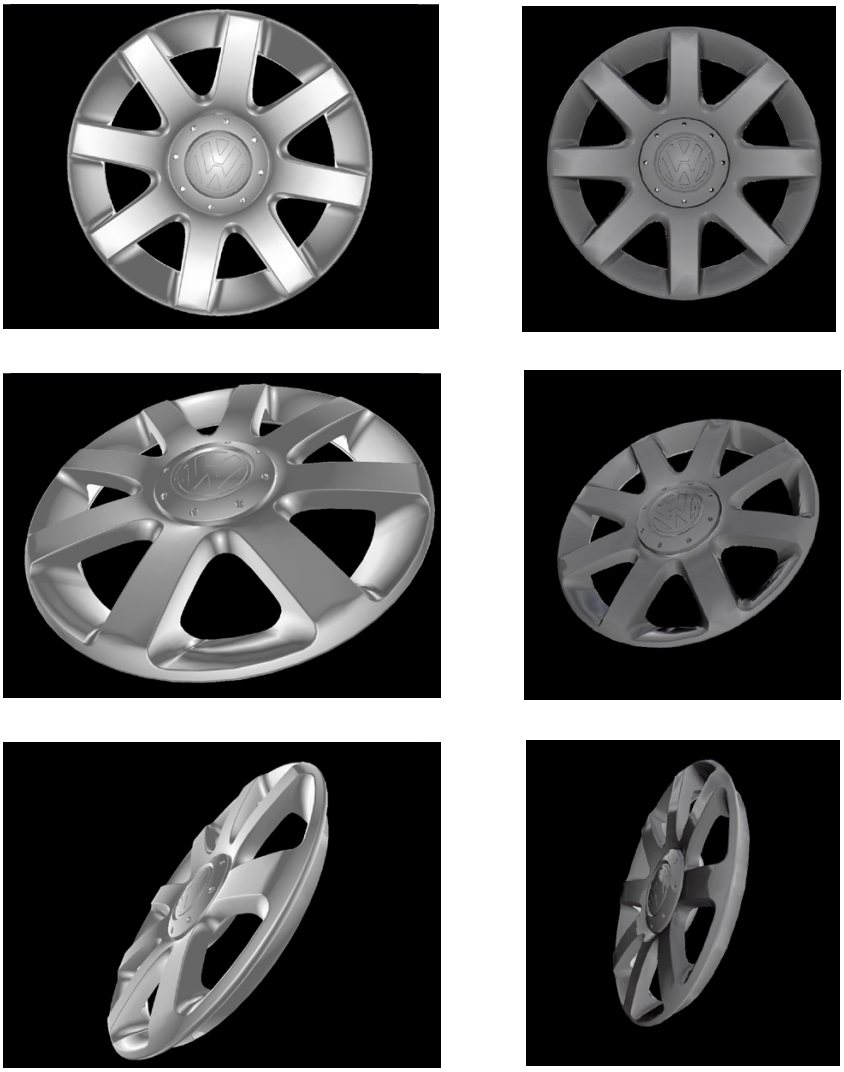


Abbildung 25: Eine Aluminium-Felge mit NDF-Shading (*links*) und einem Aluminium-Shader in VD2 (*rechts*)

Lack

Die Abbildungen 26 und 27 zeigen in der linken Spalte Screenshots des implementierten Cg-Shaders, wie er in Abschnitt 3.2.2 beschrieben ist, in der rechten Spalte ein Lack-Shader mit CLEARCOAT in VDSE/VD2.

Im wesentlichen lassen sich zwei Haupt-Charakteristika von Lack bei beiden Shadern vergleichen:

1. Die Qualität der Reflexion des Umgebung (Environment-Mapping)
2. Die Größe, Form und Spekularität des Highlights

Das Environment-Mapping des CLEARCOAT-Shaders ist recht gut: Der Verlauf der Lichtkanten auf dem Fahrzeugmodell erscheint plausibel. Jedoch entstehen bei zu spitzem Sichtwinkel Artefakte.
Beim Cg-Shader ist der Anteil dieser Environment-Reflexion in den Fresnel-Zonen etwas zu gering, müßte doch die Helligkeit in den Fresnel-Zonen deutlicher an die der spekluaren Highlights heranreichen, um überzeugen zu können.

Die Qualität des durch eine gesetzte Lichtquelle erzeugten Highlights wirkt beim Cg-Shader überzeugender als beim CLEARCOAT-Material. Hierfür ist allerdings hauptsächlich das Shading-Verfahren verantwortlich, denn während beim Cg-Shader ein Blinn-Shading implementiert wurde, verwendet das CLEARCOAT-Material das Gouraud-Shading-Verfahren[25] der Hardware.

Leider entstehen beim Cg-Shader derzeit bei der Beleuchtung teilweise Artefakte an Randflächen, wobei nicht vollständig geklärt ist, ob diese dadurch entstehen, daß Licht auf Flächen fällt, die eigentlich verdeckt sein sollen (z.B. Ritzen), oder durch etwas anderes.

[25] Beim Gouraud-Shading wird nur zwischen den Eckpunkt-Farben der Polygone interpoliert, wodurch keine langgezogenen und in der Helligkeit variierenden Highlights ermöglicht. Im Gegensatz zur Cg-Implementierung findet die Beleuchtungsberechnung also allein auf Vertex-Ebene statt.

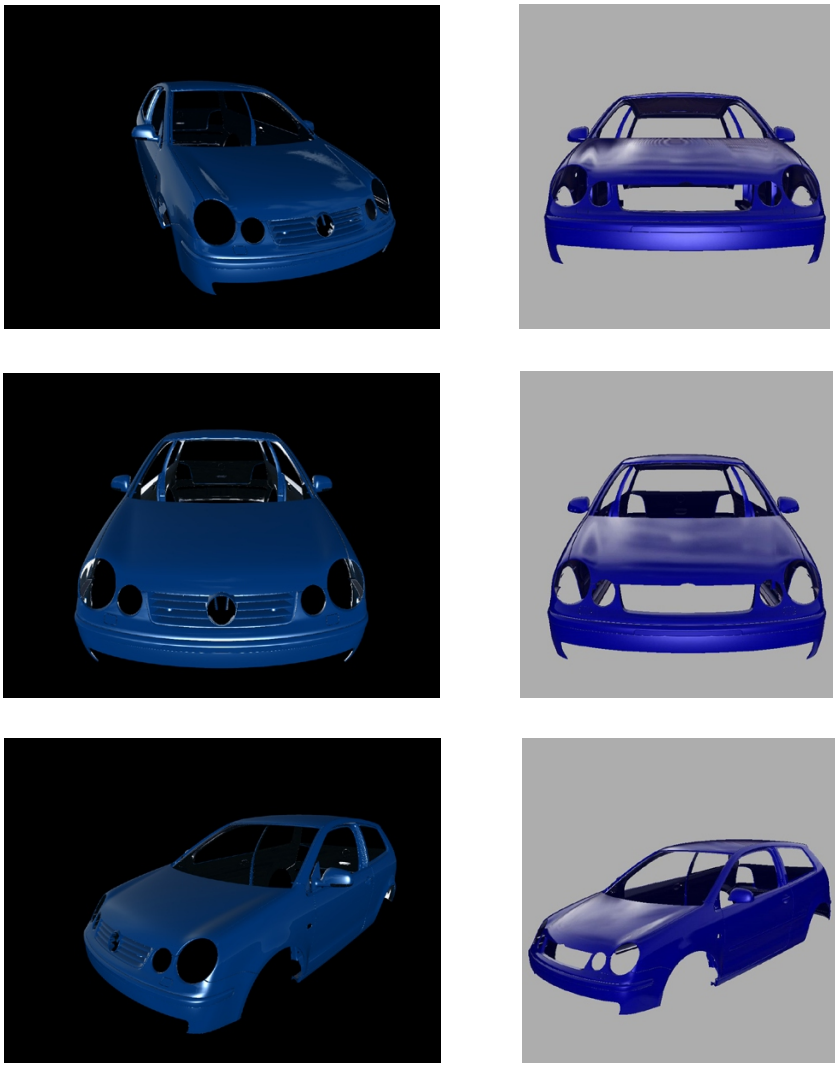


Abbildung 26: Ein Karosserie-Modell mit Lack-Shader in Cg und einem CLEARCOAT-Shader in VD2

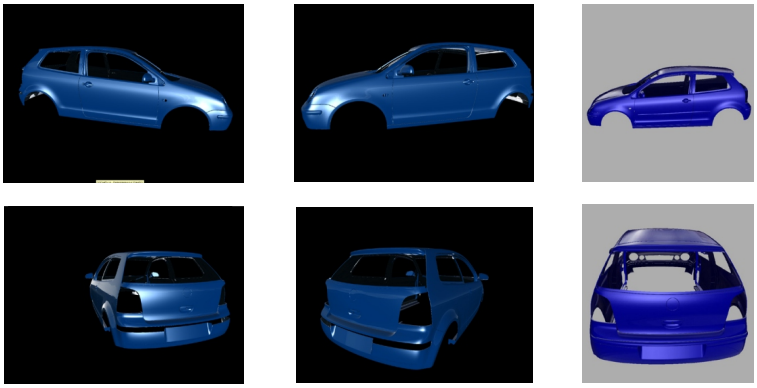


Abbildung 27: Ein Karosserie-Modell mit Lack-Shader in Cg und einem CLEARCOAT-Shader in VD2

Wie Abbildung 28 zeigt, läßt sich der Cg-Shader so modifizieren, daß Glimmerpigmente simmuliert werden. Hierzu wird eine Normal-Map verwendet, die ein leichtes „Normalen-Rauschen“ verursacht. D.h., eine Noise-Textur wird in einen Normal-Map umgewandelt und durch die Texturkoordinaten der Geometrie indiziert. Die Normalen werden wie beim Bump-Mapping dekomprimiert und in einem spekularen Beleuchtungsterm verwendet, der zu der Beleuchtung des shaders hinzu addiert wird. Der Einfluß der Glimmerpigmente auf das Aussehen des Shaders muß dabei über die Entfernung zum Betrachter gesteuert werden, so daß sie erst bei geringer Entfernung sichtbar sind.

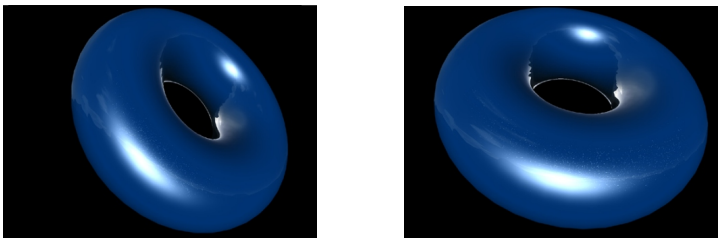


Abbildung 28: Glimmerpigmente lassen sich mit Hilfe einer Normal-Map erzeugen

5.1.3 Antialiasing

Antialiasing-Funktionen dienen der Glättung oder Reduktion von störenden Bildeffekten, wie sie zum Beispiel durch Skalierung oder bei Positionswech-

seln auftreten können. Durch Filter oder Interpolation werden diese Effekte verwischt und der Bildeindruck natürlicher.

Zusammen mit Cg-Shadern funktionieren die Antialiasing-Mechanismen der Graphik-Hardware ohne Probleme.

5.1.4 Konsequenzen für die Datenvorbereitung

Wie an den verwendeten Testmodellen bereits deutlich wurde, ist die Qualität der tesselierten Flächen für die Darstellung mit Cg-Shadern durchaus akzeptabel.
Allerdings wurden bei der Implementierung der Shader einige Kompromisse eingegangen:

Wie in Kapitel 3.2.1 beschrieben, wurden für die Transformationsmatrix zur Umrechnung der Vertices ins Texturkoordinatensystem keine aus der Geometrie abgeleiteten Tangenten und Binormalen verwendet. Zudem verfügen die verwendeten Modelle nicht über Texturkoordinaten.

Da insbesondere für die Texturierung, aber auch für pixelbasierte Beleuchtungsverfahren und Verfahren wie Bump-Mapping (bspw. zur Visualisierung von Stoffstrukturen) die Existenz von Texturkoordinaten und/oder Tangenten erforderlich ist, ist dies bei der Vorbeitung der Daten zu berücksichtigen.

Somit ließe sich auch eine Metallic-Lackierung mittels Texturen simulieren, wie in Kapitel 3.2.2 erläutert.

5.1.5 Fazit

Leider war es in dieser Arbeit bedingt durch den Testaufbau nicht möglich, verschiedene Shader in einer Szene gleichzeitig zu verwenden.

Die Anwendung einzelner Cg-Shader jedoch zeigt sich in den vorangegangenen Tests insgesamt als sowohl qualitativ gut, als auch performant. Derzeit werden recht gute Ergebnisse erzielt, allerdings ist die Entwicklung der Graphik-Prozessoren noch nicht so weit vorangeschritten, daß sich weitere wünschenswerte Verfahren/Techniken umsetzen lassen.

So sind

- Instruktions- und Funktionsumfang insbesondere der Pixelshader
- Register zum Datenaustausch zwischen Vertex- und Pixelshader
- Multipassing
- der Einsatz mehrerer Lichtquellen und Lichtquellentypen
- und Schattierungstechniken

derzeit stark begrenzt oder gar nicht vorhanden. Multipassing läßt sich zwar im Austauschformat CgFX umsetzen, birgt aber Performanzschwächen.

Durch die Verknüpfung mit der API ist die Anzahl der Lichtquellen noch immer auf acht begrenzt. Schattierungstechniken lassen sich zwar in Cg umsetzen, jedoch ist das auch hierfür erforderliche Multipassing aufgrund fehlender Rücksprung- und Schleifenfunktionen der aktuellen Graphik-Hardware bei großen Datenmengen inperformant. Abschließend läßt sich sagen, daß die Verwendung von Echtzeit-Shadern durchaus zukunftsträchtig ist, da sich die Graphik-Hardware (*GPUs*) - getrieben durch die Spieleindustrie - derzeit schneller entwickelt, als es das Gesetz von Moore[26] besagt (siehe Graphik).

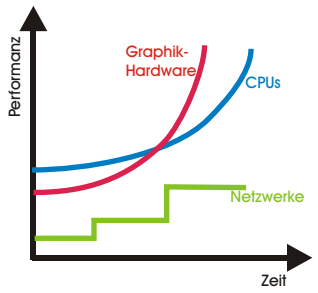


Abbildung 29: In GPUs verdoppelt sich die Anzahl der Transistoren derzeit ca. alle 6 - 12 Monate

Sowohl Leistungsfähigkeit, insbesondere der Pixelshader, als auch Funktionsumfang der Shading-Sprachen wird sich in Zukunft deutlich steigern, so daß Looping und Multipassing möglich wird. Die OpenGL-Shading-Language hat als erste bereits datenabhängige Looping-Funktionen integriert, auch wenn diese derzeit noch von keiner Hardware unterstützt werden.

5.2 Integration von Maya

Um Maya in der VR-Prozeßkette integrieren zu können, müssen zum einen die benötigten Datenformate in Maya importiert bzw. exportiert werden können, zum anderen muß sich die Verwendung von Maya beim Shading bzw. der Datenvorbereitung lohnen.

5.2.1 Schnittstellen

Für den Import bieten sich grundsätzlich die beiden Formate IGES und VRML2 an, wobei das erstere ein parametrisches Format ist, welches aus den CAD-Systemen exportiert wird. VRML2 ist dagegen ein polygonales Format, das von OPERA rausgeschrieben werden kann.

[26]Moores Gesetz: Verdopplung der Leistungsfähigkeit alle 18 Monate, Gordon Moore, Intel Corp. 1965

Die FHS-Formate[27] lassen sich zur Zeit ebenso wenig in Maya laden, wie andere CAD-Formate. Hierfür müßten Loader als Plug-In integriert werden. Die beiden folgenden Abschnitte erläutern die jeweiligen Vor- und Nachteile der Formate, bevor in einem Fazit ein Resumee gezogen wird.

Datenimport

Zum Import in Maya eignen sich einerseits das IGES-Format, das direkt in Maya geöffnet werden kann, und das VRML2-Format, das zunächst in Maya-Ascii konvertiert werden muß.

IGES Format

IGES ist ein parametrisches Datenformat, das beispielsweise aus CATIA exportiert wird. Da Maya mit NURBS umgehen kann, lassen sich IGES-Daten problemlos laden und anzeigen.

Für die VR ist es jedoch wichtig, daß die Daten in tesselierter Form als Dreiecksnetz vorliegen (seihe Abschnitt 1.2.1).
Beim Import von parametrischen Daten müssen also zunächst

- die topologiebasierte Tesselierung,
- das Verschmelzen entstandener Löcher (Healing)
- und die Optimierung

in Maya erfolgen. In der Prozeßkette werden diese Aufgaben von OPERA übernommen.
Erst nach der Tesselierung kann Maya für das Shading genutzt werden.

Die Umwandlung der parametrischen Daten ist recht aufwändig: Zunächst werden alle NURBS-Flächen ausgewählt und tesseliert[28].

Das Ergebnis sind den einzelnen NURBS-Flächen entsprechende, nicht zusammenhängende Polygonflächen. Aufgrund der individuellen Topologie der parametrischen Oberflächen, fällt auch die Tesselierung unterschiedlich aus, was dazu führt, daß zwischen Kanten nebeneinanderliegender Flächen Löcher entstehen (Abbildung 30).

Diese Löcher zu verschließen erfordert einen erheblichen Arbeitsaufwand, müssen die Flächen doch durch Anlegen neuer Polygone verbunden werden.

Die verwendeten parametrischen Flächen sind zudem meist getrimmte NURBS-Flächen. Getrimmte NURBS-Flächen sind Flächen mit orientierten Flächenkurven, wobei diese in der Parameterbene als NURBS-Kurve bezüglich eines Parameterintervalls dargestellt werden. Deshalb lassen sich

[27] FHS und FHB, wobei letzteres ein Binary-Format ist
[28] über MODIFY -▷ CONVERT -▷ NURBS TO POLYGONS

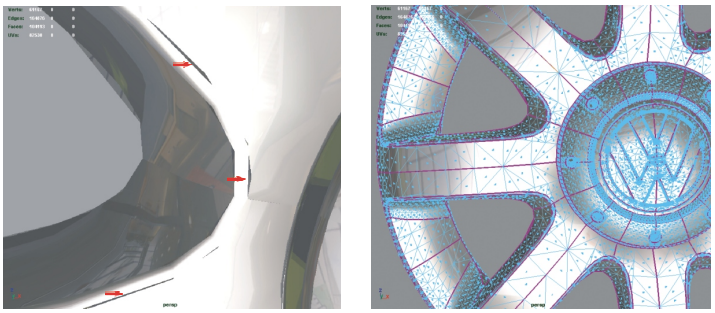


Abbildung 30: Löcher zwischen einzelnen Polygonnetzen sind unvermeidbar, da nicht zusammenhängende NURBS-Flächen aufgrund ihrer Topologie unterschiedlich tesseliert werden

diese Flächen nicht bereits vor der Tesselierung zusammenfügen.

Eine unzusammenhängende Polygonstruktur und die damit verbundene manuelle Aufbereitung der Daten sind beim Import von IGES-Daten in Maya also unvermeidbar.

VRML2 Format

Damit Daten im VRML-Format geladen werden können, müssen diese zunächst über einen Kommandozeilen-Konverter ins Maya-Ascii-Format konvertiert werden. Dieses läßt sich dann in Maya öffnen.

Bei der Konvertierung gehen die Hierarchie-Strukturen der VRML-Daten jedoch offenbar verloren, denn importierte Bauteile bleiben nicht als Objekte erhalten, sondern zerfallen in einzelne Teilflächen. Dadurch sind sehr viele Polygon-Kanten und -Eckpunkte mehrfach vorhanden. Diese Inkonsistenz in der Oberflächenstruktur hat mehrere Nachteile:

- Eine erhebliche Anzahl Daten wird mehrfach gespeichert
- Informationen über Flächen- und Punktnormalen gehen evtl. verloren, so daß statt eines Gouraud- oder Phong- Shading nur ein Flat-Shading möglich ist
- Die Berechnung konsistenter Texturkoordinaten ist nicht möglich

Bei einer Bereinigung der Daten müßten zunächst alle einzelnen Flächen zu einem geschlossenen Dreiecksnetz zusammengefügt und anschließend alle mehrfach vorhandenen Kanten und Vertices eliminiert werden. Anschließend wären die Flächen- bzw. Punktnormalen wieder entsprechend auszurichten oder neu zu setzen.

Ein Shading ist erst nach dieser aufwendigen Aufbereitung möglich.

Datenexport

Als Exportformat kommen prinzipiell nur die polygonalen Formate VRML2 und FHS/FHB in Frage, da nur diese vom VR-System importiert werden können.
Da Maya derzeit nicht über einen Exporter für FHS-Daten verfügt, ist VRML2 das einzig verfügbare Export-Format.

Die Qualität der exportierten Daten ist gut: Bei der Darstellung in einem VR-System gibt es keine weiteren Komplikationen. Auch in Maya angelegte Texturkoordinaten werden mit der zugehörigen Textur problemlos rausgeschrieben.

Allerdings ist weder im VRML2-Format, noch im FHS/FHB-Format derzeit die Integration von CgFX-Materialien vorgesehen. Das bedeutet, daß in VRML2 zwar mit dem Befehl

```
appearance Appearance
{
material DEF cgfxShader2.0
}
```

die Materialdefinition kenntlich gemacht wird, jedoch werden weder der Name der CgFX-Datei, noch eine Referenz auf die zugehörige Shader-Datei oder gar eingestellte Parameter mit exportiert.

Aus Maya werden aufgelegte CgFX-Shader mit dort eingestellten Attributen derzeit also nicht in einem für das VR-System lesbaren Format exportiert.

5.2.2 Das CgFX Plug-In

Um das CgFX Plug-In und die erweiternden Funktionalitäten beurteilen zu können, werden im folgenden einzelne Funktionen des Plug-Ins bzw. seiner Erweiterungen durch MEL-Skripte näher betrachtet.

Übersetzen eines Shading Network

Maya bietet hervorragende und intuitive Möglichkeiten zum Shading und zur Texturierung virtueller Modelle. Deshalb ist eine Funktion, die vorhandene Shading Networks aus Maya in echtzeitfähige CgFX-Shader übersetzt, prinzipiell ein guter Ansatz.

Leider ist jedoch die Qualität der Konvertierung derzeit noch nicht optimal. So ist die Konvertierungsfunktion derzeit auf einfache Shading Net-

works beschränkt und besitzt einige Restriktionen.
Es werden

- nur Blinn, Phong, Phong E. und Lambert-Materialien,
- nur ambiente, Punkt- und Umgebungslichhtquellen,
- Spotlichter als Punktlichter,
- keine Transparenzen,
- keine Schatten,
- nicht alle spekularen Attribute,
- keine Reflexionen,
- keine multiplen UV-Sets
- Texturen im REPEAT-Modus

übersetzt.
Andere Materialien-Typen wie Ebenen-Shader oder anisotropische Materialien werden bei der Konvertierung einfach übersprungen.

Texturen müssen in einem übersetzbaren Format (z.B. TGA) vorliegen und quadratisch sein. Die Auflösung muß der einer Zweier-Potenz entsprechen. Die Anzahl der verwendeten Lichtquellen in einer Szene und die Attributwerte müssen vor der Konvertierung festgelegt werden und lassen sich anschließend nur durch Manipulationen im Code selbst korrigieren.

Bei der Konvertierung werden Texturen, wie in Abschnitt 4.2.2 bereits erwähnt, im DDS-Format abgespeichert. Übersetzte Shader, die mit Texturen arbeiten, werden demnach nur mit der DirectX-API korrekt geladen. Zudem werden nur 2D-Texturen berücksichtigt; Environment-Mapping mittels Cube-Mapping wird folglich nicht in CgFX umgesetzt.

Abbildung 31 zeigt ein Blinn-Material, wie es von Maya im Offline-Modus gerendert wird, und wie das übersetzte Material als CgFX-Shader im WorkSpace dargestellt wird. Der direkte Vergleich zeigt kleine Unterschiede in Größe und Form des Highlights, im wesentlichen aber ist die Umsetzung des originalen Maya-Materials korrekt.

Modifikation vorhandener Shader

Vorhandene CgFX-Shader -ob mitgelieferte aus der Demo-Bibliothek, oder selbst konvertierte- können durch Slider im AttributeEditor verändert werden. Änderungen der Parameterwerte können im WorkSpace direkt mitverfolgt werden.

Leider besteht derzeit keine Möglichkeit, die veränderten Attributwerte

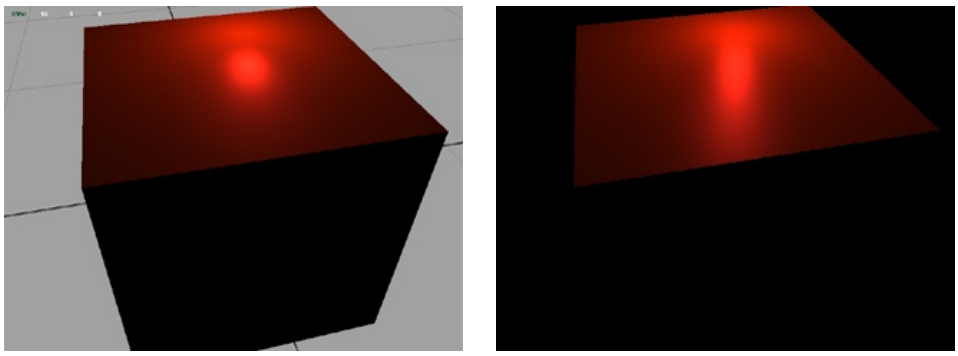


Abbildung 31: *links:* Das Blinn-Material, nach CgFX übersetzt, wird im WorkSpace angezeigt, *rechts:* Dasselbe Blinn-Material, von Maya gerendert

wieder mit dem CgFX-Code abzuspeichern, um so die vorhandene Shader-Bibliothek möglicherweise zu erweitern.

Die einzige Möglichkeit bietet die Editier-Funktion „CgEd“: In einem Editor wird der CgFX-Code geöffnet und kann direkt manipuliert werden. Der manipulierte Code kann natürlich auf diese Weise auch abgespeichert werden.

Verknüpfung mit Lichtquellen

Wird ein CgFX-Shader in einer Szene verwendet, wird er nicht automatisch mit einer oder mehreren Lichtquellen verknüpft. Manche Shader aus der Demo-Bibliothek besitzen für eine Lichtquellenanbindung keine Slots, andere maximal zwei. Die Slots werden in der Attributeliste des Shaders aufgeführt und können dort mit in Maya gesetzten Lichtquellen verbunden werden.

Damit der Shader auch auf Änderungen der Lichtquelle entsprechend reagieren kann - bspw. Änderungen der Lichtfarbe oder - intensität - müssen auch die entsprechenden Attribute verbunden werden. Dies kann entweder manuell im ConnectionEditor erfolgen, oder über das MEL-Skript „ltcn“ automatisiert werden. Auch hier gilt: Es können nur diejenigen Attribute verknüpft werden, die sowohl bei der Lichtquelle, als auch beim CgFX-Shader vorhanden sind.

5.2.3 Maya in der Prozeßkette

Maya ließe sich prinzipiell an zwei Stellen im Datenvorbereitungs-Prozeß integrieren:

1. Als Ersatz für OPERA
2. Zwischen OPERA und dem VR-System VDSE/VD2

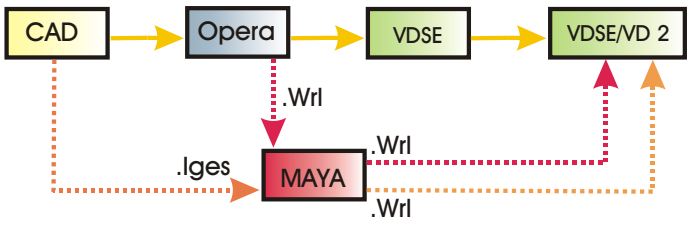


Abbildung 32: Maya in der Prozeßkette

Auf OPERA zu verzichten, bedeutet, Tesselierung, Optimierung, sowie Shading und Texturierung in Maya durchzuführen.
Der Vorteil besteht darin, auf ein Aufbereitungs-Tool verzichten zu können. Dadurch würden Wartungskosten und Lizenzgebühren für OPERA gespart. Der große Nachteil ist der Verzicht auf Automatismen bei Tesselierung und Optimierung, die momentan durch OPERA im Batch-Modus ablaufen können. Die manuelle Datenaufbereitung dieser Art ist demgegenüber äußerst zeitaufwendig.

Maya zwischen OPERA und dem VR-System zu platzieren beschränkt den genutzten Funktionsumfang auf das Shading, gegebenenfalls lassen sich in Maya erstellte Animationen exportieren. Mayas komfortable Oberfläche läßt sich benutzerorientiert einstellen und bietet umfangreiche Shading- und Texturierungsfunktionen.
Problematisch sind wie in Abschnitt 5.21 erläutert, die Schnittstellen: Der Import von VRML2-Daten ist nicht zufriedenstellend, zudem werden keine CgFX-Shader mit exportiert. Das Fraunhofer Datenformat wird von Maya bisher nicht unterstützt.

5.2.4 Fazit

Wie die Untersuchungen aus den vorangegangenen Abschnitten zeigen, ist die Übersetzungsfunktion des CgFX-Plug-Ins noch nicht so weit entwickelt, daß komplexe Shading Networks übersetzt werden können. Viele Materialien werden unzureichend oder gar nicht konvertiert, so daß die gezielte Implementierung einzelner Shader sowohl qualitativ hochwertigere, als auch performantere Ergebnisse liefern würde.

Qualitativ besser, da Verfahren wie Cube-Mapping, Bump-Mapping und

andere Algorithmen umgesetzt werden können, die mit der Konvertierungsfunktion nicht möglich sind, jedoch eine verbesserte visuelle Darstellung ermöglichen.

Performanter, da Register für den Datenaustausch zwischen Vertex- und Pixel-Shader besser ausgenutzt werden können.
Aufgrund der Tatsachen, daß

- die Import-Schnittstelle für VRML2-Daten in Maya keine befriedigenden Ergebnisse erzielt,
- insbesondere die Konvertierungsfunktion des CgFX-Plug-Ins in Maya bisher zu wenige Möglichkeiten bietet, komplexere Effekte in echtzeitfähige CgFX-Shader zu übersetzen
- und derzeit kein Format für Maya existiert, daß Referenzen auf bestimmte CgFX-Shader mit Attributen exportiert und in das VR-System VD2 importiert werden kann,

ist die Integration von Maya als Werkzeug für Shading und Texturierung in die Prozeßkette zum jetztigen Zeitpunkt nicht empfehlenswert.

Eine Cg- oder sogar CgFX-Schnittstelle zu einem VR-System bietet demgegenüber den Vorteil, daß kein Exportformat geschaffen werden müßte, da die Shader im System selbst angezeigt würden.
Das CgFX-Format bringt dabei den Vorteil der Multipass-Effekte mit, ist jedoch momentan nur mit der DirectX-API verwendbar aufgrund des geforderten Texturformates DDS.
Cg-Shader bieten bisher noch keine Multipassing-Funktion, haben im Praxistest allerdings ihre Echzeitfähigkeit in einem VR-System unter Linux bewiesen (siehe Abschnitt 5.1).

6 Ausblick

Die in dieser Arbeit untersuchten Fragestellungen, gefundene Lösungsansätze und Implementierungen werfen wiederum weiterführende Fragen und Aufgabenstellungen auf.

Zur Integration von Maya als Shading-Tool in der Prozeßkette wären beispielsweise Schnittstellen und evtl. auch Datenformate zu entwickeln, die

- den Import polygonaler Daten aus Opera und
- den Export polgonaler Daten inklusive Echtzeit-Shadern ermöglichen

Hierfür eignet sich zum einen das interne Arbeitsformt von OPERA, das CSB-Format, für den Import, für den Export wäre zu untersuchen, inwieweit sich das VRML2-Format oder das FHS-Format eignet bzw. erweitern ließe.

Die Anbindung von Cg- oder CgFX-Shadern an OPERA oder an VD-SE/VD2 ist ebenfalls ein Weg, der weiter verfolgt werden könnte.

Wenn Echtzeit-Shader in Zukunft genutzt werden sollen, muß eine Bibliothek an Shadern zur Verfügung stehen. Eine solche Bibliothek zu entwickeln, eröffnet wiederum einen Aufgabenbereich. Um den Szenenrealismus zu erhöhen, insbesondere im Fahrzeuginterieur, wären Shader zur Visualisierung von

- Chrom
- Holz
- Fensterglas
- Sitzbezügen aus Stoff und Leder
- Lichtabstrahlungen z.B. von Rückleuchten, aber auch im Cockpit
- Schatten
- u.a.

zu entwicklen.
Für Stoffe und Leder eignen sich bspw. Bump-Mapping-Verfahren, evtl. auch die bei HP[29] entwickelten Polynomial Texture Maps, die es ermöglichen, ausschließlich mit Hilfe von Texturen, strukturierte Oberflächen zu simulieren, die hochgradig photorealistisch sind [20].
Fensterglas, aus dem Fahrzeuginterieur heraus betrachtet, ließe sich durch Refraktion mit Hilfe einer Environment-Map erzielen, Schatten lassen sich mit Hilfe von Shadow-Maps rendern.

[29] Hewlett-Packard

In Zukunft wird sich die Graphik-Hardware weiter entwickeln, so daß noch komplexere Algorithmen auf der Hardware umgesetzt werden können. Welche Shading-Sprache dazu am besten geeignet ist, bleibt abzuwarten. Sicher jedoch ist, daß die Shading-Sprachen konvergieren und sich zukünftig kaum unterscheiden werden. In der Verwendung von Echtzeit-Shadern liegt jedoch ein enormes Potential!

Literatur

[1] T.Loeder C.Berndt. *Maya 4.5 und mental ray for Maya*. Galileo Press GmbH, Bonn, 2002.

[2] Sebastien Domine. Using Texture Compression in OpenGL.

[3] Wolfgang F. Engel. *Effiziente Echtzeit-Beleuchtungsalgorithmen*, November 2003.

[4] Wolfgang F. Engel. *ShaderX2: Shader Programming, Tips and Tricks with DirectX 9.0*. Wordware Inc., September 2003.

[5] Dr. U. Hartwig. *Journalisten und Wissenschaftler im Gespräch: Faszination Farbe*. BASFCoatings AG, Mannheim, September 1999. www.basf-coatings.de.

[6] H.-P. Seidel J. Kautz, K. Daubert. User-Defined Shading Models for VR Applications. Technical report, Max-Planck-Institut, Saarbrücken, Januar 2002.

[7] W.R Mark K. Proudfoot et al. A Real-Time Procedural Shading System for Programmable Graphics Hardware. *Proceedings SIGGRAPH*, 2001.

[8] E.M. Sparrow K.Torrance. Theory for Off-Specular Reflection from Roughened Surfaces. *Journal of the Optical Society of America*, pages 57(9):1105–1114, 1967.

[9] M. Olano M.S. Peercy et al. Interactive Multi-Pass Programmable Shading.

[10] nVidia. http://developer.nvidia.com/object/MayaCgPlugin.html.

[11] nVidia. Cg Toolkit- User‘s Manual. A Developer‘s Guide to Programmable Graphics, Februar 2003.

[12] OpenSG. http://www.opensg.org.

[13] OpenSGPLUS. http://www.opensg.org/OpenSGPLUS/index.EN.html.

[14] E. Ferley P. Dumont-Becle et al. Multi-Texturing Approach for Paint Appearance Simulation on Virtual Vehicles. *DSC 2001 Sophia Antipolis*, September 2001.

[15] B.-T. Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, pages 18(6): 311–317, June 1975.

[16] K.Torrance R. Cook. A Reflectance Model for Computer Graphics. *Proceedings SIGGRAPH*, pages 307–316, August 1981.

[17] M.J. Kilgard R. Fernando. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, Februar 2003.

[18] C. Schlick. An inexpensive BDRF Model fpr physically based Rendering. *Eurographics '94*, pages 149–162, September 1994.

[19] E. Haines T. Moeller. *Real-Time Rendering*. AK Peter Ltd, Februar 1999.

[20] Hans Wolters Tom Malzbender, Dan Gelb. *Polynomial Texture Maps*, 2001. http://www.hpl.hp.com/research/ptm.

[21] H.-P. Seidel W. Heidrich. Realistic, Hardware-accelerated Shading and Lighting. *Proceedings SIGGRAPH*, pages 171–178, August 1999.