shaded

Studienarbeit 2
Shader Editor
Hochschule für Technik Rapperswil HSR
Sommersemester 2006

Student: Christian Oberholzer
Betreuer: Dr. Urs-Martin Künzi

Inhaltsverzeichnis

Teil A - Einführung

Kapitel I: Abstract....2

Kapitel II: Aufgabenstellung....3

1. Einführung....3
2. Aufgabe....3
3. Geforderte Berichtsinhalte:....4
4. Hinweise....4
5. Termine....4
6. Betreuung....4

Kapitel III: Management Summary....6

1. Ausgangslage....6
2. Durchführung der Arbeit....8

Teil B - Projektmanagement

Kapitel I: Projektplan....12

1. Einführung....12
2. Projektübersicht....12
3. Projektphasen und Meilensteine....13
4. Prototypen....15
5. Artefakte....15
6. Risikoanalyse....16
7. Qualitätsmassnahmen....16

Kapitel II: Anforderungsspezifikation....18

1. Vision....18
2. Aktoren....19
3. Use-Case Modell....20
4. Annahmen und Abhängigkeiten....25
5. Anforderungen....25

Kapitel III: Schlussbericht Projektmanagement....29

1. Planung....29
2. Aufwandsschätzungen....29

Teil C - Technischer Bericht

Kapitel I: Shaderprogrammierung....32

1. Übersicht....32
2. Ursprung....32
3. Die Rendering Pipeline....33
4. Vertex- und Pixelshader....34

5. Shader-Sprachen....34

Kapitel II: Domain Analyse....36

1. Einführung....36
2. Shadergraph....36
3. Domain Modell....37

Kapitel III: Übersicht Design....43

1. Einführung....43
2. Logische Architektur....43
3. Die wichtigsten Elemente....44

Kapitel IV: shaded Kern....51

1. Einführung....51
2. Logging....51
3. Mathematik-Klassen....52

Kapitel V: User Interface....54

1. Einführung....54
2. Analyse verwandter Produkte....54
3. Entwurf....55
4. Design....59

Kapitel VI: Scene....62

1. Einführung....62
2. Lichtquellen....63
3. Modelle....63
4. Geometrie....63
5. Materialien....64

Kapitel VII: Shadergraph Plugin....65

1. Übersicht....65
2. Design....65

Kapitel VIII: Umsetzung....74

1. Assemblies....74
2. Screenshots der Applikation....74
3. Interessante Fakten....76

Kapitel IX: Test....77

1. Test Benutzeroberfläche....77
2. Test Shadergraph-Plugin....77
3. Funktionale Anforderungen....78

Kapitel X: Schlussfolgerungen....80

1. Zusammenfassung....80
2. Erreichte Ziele....80
3. Ausblick....80

Teil D - Anhang

Anhang I: Glossar....84

Anhang II: Literaturverzeichnis....85

Anhang III: Persönlicher Bericht....86

Anhang IV: Dank....87

Anhang V: Programmierrichtlinien....88

Anhang VI: Sitzungsprotokolle....90

Anhang VII: Zeitplanung....95

Teil A - Einführung

Kapitel I: Abstract

1988 führte Pixar einen offenen Standard zum Rendern von 3-Dimensionalen Szenen ein; die RISpec („The RenderMan Interface Specification"). Teil dieser Spezifikation war die so genannte „shading language" (SL). Sie ermöglichte es, Oberflächen mit beliebigen programmierbaren Algorithmen zu beschreiben, statt wie bisher über ein eingeschränktes Set von Parametern und Konstanten. Dieser neue Standard feierte in den darauf folgenden Jahren grosse Erfolge und ist heute in der Filmproduktion weit verbreitet. Mit der Einführung von DirectX8 wurde der Ansatz der Shader (in einer Shading Language geschriebene Programme) auch auf den Bereich des Echtzeitrenderings übertragen. Moderne Grafikkarten besitzen heutzutage programmierbare Prozessoren über welche beliebige Effekte realisiert werden können.

Da diese Shader für die visuelle Erscheinung und Qualität eines Produktes verantwortlich sind sollten Grafiker deren Programmierung übernehmen. Grafiker sind sich jedoch nicht gewöhnt Programme zu schreiben und müssten folglich dafür geschult werden.

Die vorliegende Studienarbeit versucht dieses Problem zu lösen, indem es Shader in Form einer Graphenstruktur visualisiert. Solche Strukturen werden bereits von verschiedenen Produkten verwendet. Im speziellen sind das die weit verbreiteten Modellierungsprogramme Blender und Cinema4D. Dadurch sind sich viele Grafiker bereits daran gewöhnt. Shaded generiert aus solchen Graphen Shader und stellt sie in einer Vorschau-Ansicht dar. Damit wird es für Grafiker möglich, auf einfache Art und Weise Shader zu erstellen, ohne dass sie das Programmieren lernen müssen.

Kapitel II: Aufgabenstellung

Studienarbeit SS06: „Shader Editor“

Student: Christian Oberholzer

1 Einführung

Um den Produzenten von Computerspielen und anderen Echtzeit Grafikapplikationen eine maximale Freiheit bei der Gestaltung der visuellen Erscheinung ihres Produktes zu gewährleisten sind moderne Grafikkarten, über so genannte Shader, programmierbar. Nachdem das Instruktionsset der ersten programmierbaren Grafikkarten noch sehr klein war, können auf neuen Grafikkarten auch komplexe Programme mit Schlaufen und Bedingungen ausgeführt werden.

Obwohl solche neue Hardware grosse Freiheiten und beinahe fotorealistische Resultate liefern kann bringt sie auch einen grossen Nachteil mit sich. Shader sind verantwortlich für das Rendering, und damit für die die Erscheinung und visuelle Qualität von 3-Dimensionalen Modellen. Daher wäre es am besten, wenn die Grafiker, welche auch die Modelle erstellt haben, die entsprechenden Programme schreiben könnten. Es liegt auf der Hand, dass Grafiker zur Erstellung von Shadern manchmal die Unterstützung eines Programmierers benötigen. Dies führt zu einer weiteren, nicht unbedingt nötigen Belastung der, bei Computerspielen oft bereits überbelasteten, Entwickler. Für dieses Problem soll eine Lösung implementiert werden.

2 Aufgabe

Ausgehend von der in der Einführung erwähnten Problemstellung soll ein Programm geschrieben werden, welches die Grafiker bei der Erstellung von Shadern unterstützt. Die Lösung muss nicht von Grund auf neu erfunden werden, sondern kann sich an folgenden Produkten orientieren:

- rtzen RT/shader: Ein Produkt der unterdessen nicht mehr existierenden Firma rtzen.
- nVidia FX Composer: http://developer.nvidia.com/object/fx_composer_home.html
- ATI Render Monkey: http://www.ati.com/developer/rendermonkey/index.html

Das Resultat der Arbeit soll eine demonstrierbare Software auf Stufe Prototyp („Proof of Concept“) sein, ergänzt mit einer Dokumentation, die wichtige Überlegungen der Analyse, des Entwurfs und der Implementierung festhält.

Die folgenden Ziele sind zu erfüllen:

- Programme sollen durch einen Graphen (ähnlich wie in RT/shader) visualisiert und erstellt werden können. Es ist ein entsprechendes Steuerelement zu implementieren.
- Der generierte Code muss vom Programm angezeigt und exportiert werden können.
- Der editierte Shader soll in einem Vorschau-Fenster laufend angezeigt und aktualisiert werden, so dass die Auswirkung von Änderungen im Programm sofort sichtbar sind.
- Für dieses Vorschau-Fenster müssen Resourcen, das heisst Texturen und Modelle, geladen werden können.
- Durch die rasante Entwicklung der Grafikkarten und die Rivalität zwischen OpenGL und DirectX existieren verschiedene Versionen von Shadersprachen. Die für den resultierenden Prototypen verwendete Version kann frei gewählt werden. Es sollte jedoch beachtet werden, dass das Programm erweiterbar ist, so dass zu einem späteren Zeitpunkt auch andere Versionen unterstützt werden können.
- Das resultierende Programm kann in einem beliebigen Format gespeichert werden. Es sollte jedoch möglich sein über Plugins andere Ausgabeformate zu implementieren.

Die entwickelte Software sollte zusätzlich auf einem Windows Rechner mit den gegebenen minimalen Hardware-Anforderungen installiert werden, sowie bei Verwendung der dokumentierten Entwicklungs-Tools auch generiert werden können. Es ist eine Installationsbeschreibung bereitzustellen. Abzugeben sind zwei CD-ROMs (MS Windows kompatibel), die die Software und den Bericht in elektronischer Form enthalten (PDF-Dateien und Originaldateien des benutzten Textsystems). Der Bericht ist zusätzlich zweimal in gedruckter Form mit Ringbindung (kein Ordner!) bereitzustellen. Basis für die Bewertung der Arbeit stellen der gedruckte Bericht, sowie die auf der CD bereitgestellte Software dar.

3 Geforderte Berichtsinhalte:

- **Resultatdokumentation:** Sie beschreibt alle Resultate der Arbeit und soll projektbegleitend gemäss dem für die Arbeit gewählten Vorgehensmodell erstellt werden. Sie soll alle wichtigen Informationen enthalten, die ein Ingenieur benötigt, der die Arbeitsresultate ohne vorgängige Kenntnis des Projekts weiterverwenden möchte.
- **Projektdokumentation:** Sie umfasst jegliche Dokumentation, die sich auf die Durchführung der Studienarbeit bezieht. Dazu gehören auch ein nachgeführter Projektplan (Arbeitspakete, Zeitplan mit Meilensteinen, Plan- und Ist-Aufwände), Kurzprotokolle aller Besprechungen und ein Projektschlussbericht (was wurde erreicht bzw. nicht erreicht, was ist in der Durchführung gut/schlecht gelaufen, Schlussfolgerungen).

Im Weiteren soll die Gliederung und der Inhalt des Berichts den Regelungen der Abt. Informatik entsprechen (siehe Richtlinien unter den Web-Seiten der Abt. Informatik: http://i.hsr.ch, Menü „Studienarbeiten->DokuAnleitung“). Teildokumente sollen derart in den Bericht integriert werden, dass ein hierarchisches Gesamt-Inhaltsverzeichnis mit durchgängiger Seitennummerierung ermöglicht wird.

Datenblätter, Handbücher und allfällige Kopien anderer Dokumente, die nicht selbst erstellt wurden, aber für weitere Arbeiten mit den Systemen nützlich sind, sollen separat zum Gesamtbericht abgegeben werden (Abgabeform frei; evtl. auch nur auf Abgabe-CD). Berichtsinhalte, die nicht selbst erarbeitet wurden, sind mit ihrer Quelle zu bezeichnen. Hinweise zum korrekten Zitieren sind unter folgenden URLs zu finden:

- http://www.plagiarism.org/research_site/e_citation.html
- http://www.plagiarism.org/research_site/e_what_is_plagiarism.html

4 Hinweise

Es wird eine Durchführung der Arbeit nach den Grundsätzen des Software Engineerings erwartet. Dazu gehört eine Projektplanung mit Sollwerten, eine Erfassung der Istwerte und eine abschliessende Beurteilung des Projektverlaufs im Soll-/Istvergleich. Insbesondere sind terminliche und aufwandmässige Planabweichungen zu dokumentieren und begründen.

5 Termine

Datum	Anlass
3.4.2006	Arbeitsbeginn
7.7.2006	Abgabe der Kurzbeschreibung per Email an Abteilungssekretariat Frau J. Ebnöther (jebnoeth@hsr.ch). Es ist das Formular unter http://www.hsr.ch/downloads_ext/kurzfass.doc zu benützen.
7.7.2006	Abgabe des Berichts an den Betreuer.

6 Betreuung

Betreuer: Dr. Urs-Martin Künzi, Email: urs-martin.kuenzi@hsr.ch

Während der Durchführung der Arbeit finden regelmässig, etwa alle 2 Wochen, Besprechungen mit dem

Betreuer statt. Die genauen Termine werden jeweils vorgängig per Mail abgesprochen.

Rapperswil, den 27.3.2006

Urs-Martin Künzi

Kapitel III: Management Summary

1 Ausgangslage

1.1 Ursprung

Traditionellerweise wurden beim Rendern von 3-Dimensionalen Szenen die Oberflächen durch ein gegebenes Set von Parametern und Konstanten beschrieben. Die Modellierungssoftware verwendete dann die gegebenen Parameter als Eingabewerte für fest programmierte Algorithmen. Dadurch wurden Grafiker in ihrer gestalterischen Möglichkeit eingeschränkt und verschiedene Produktionen erhielten durch die Verwendung der gleichen Algorithmen ein ähnliches Aussehen. Seit der Einführung der RISpec („The RenderMan Interface Specification“) durch Pixar im Jahr 1988 wurde es über eine sogenannte „shading language“ (kurz SL) möglich, diese fixen Algorithmen durch frei programmierbare eigene Algorithmen zu ersetzen.

Mit der Einführung von Microsofts DirectX8 im Jahr 2000 wurde die Idee der freien Programmierbarkeit über sogenannte Vertex- und Pixelshader auch auf den Bereich der Echtzeitgrafik übertragen. Diese Vertex- und Pixelshader besassen zuerst nur sehr beschränkte Möglichkeiten. Die neuste Generation von Grafikkarten kann jedoch sehr komplexe Programme ausführen.

1.2 Problemstellung

Durch die Einführung von Shadern in der Echtzeitumgebung entstand ein Dilemma. Einerseits bestimmen die Programme die visuelle Erscheinung und Qualität eines Produktes und sollten damit eine Aufgabe der Grafiker sein. Andererseits ist die Programmierung von solchen Programmen nicht einfach und daher werden oft Programmierer zur Unterstützung der Grafiker benötigt. Es soll eine Lösung gefunden werden, mit welcher dieses Dilemma gelöst und die Erstellung von Shadern alleine den Grafikern überlassen werden kann.

1.3 Ziel

Die Studienarbeit soll das gegebene Problem lösen, indem der Benutzer über eine Graphenstruktur visuell programmieren kann. Diese Idee wurde bereits erfolgreich in Modellierungssoftware wie zum Beispiel Blender oder Cinema4D umgesetzt. Zusätzlich sollen einige komplexere Konzepte der Shaderprogrammierung durch die Graphenstruktur abstrahiert werden, so dass sich der Benutzer nicht mehr, oder nur noch beschränkt, darum kümmern muss. Die Shader sollen in einer Vorschau-Ansicht angezeigt werden, so dass Änderungen des Programmes sofort sichtbar werden. Durch diese Lösung soll es Grafikern ermöglicht werden Shader ohne mithilfe von Software-Entwicklern zu erstellen. Abbildung 1 zeigt die Interaktion zwischen dem Benutzer und shaded.

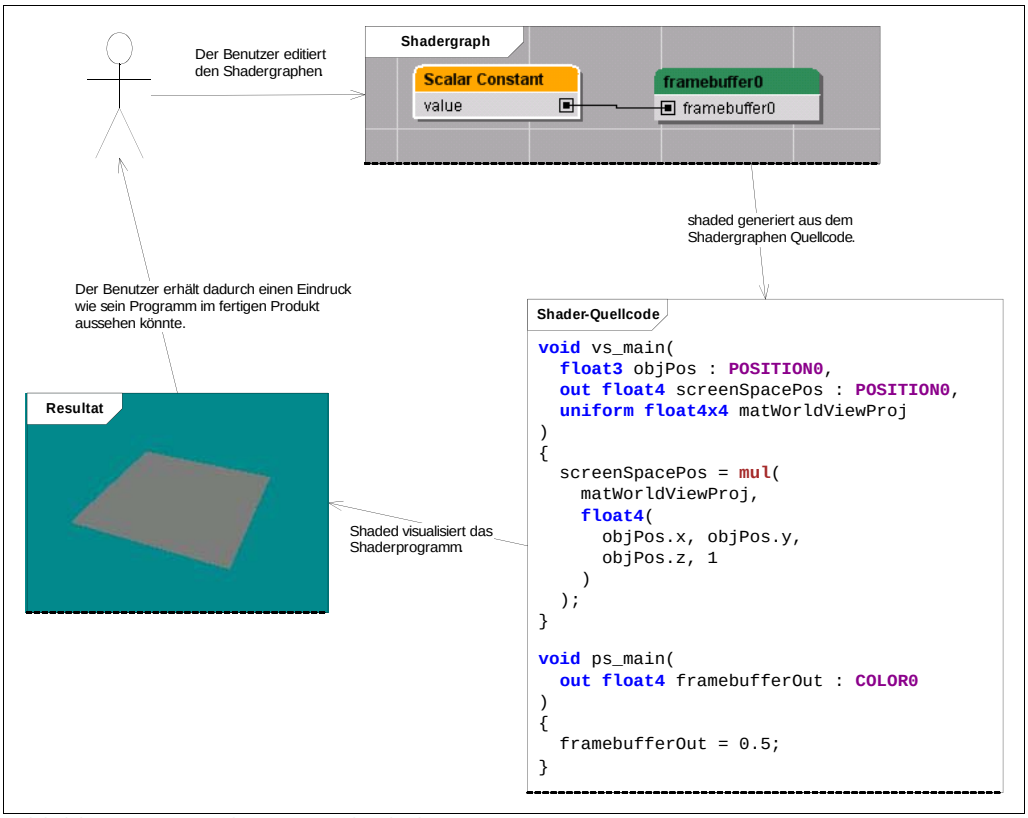


Abbildung 1: Interaktion mit shaded

1.4 Ähnliche Produkte

Es existiert eine Reihe von Produkten welche ebenfalls versuchen die Erstellung von Shadern zu vereinfachen. Diese Produkte werden im folgenden aufgeführt:

a) nVidia FX Composer [NVID01]: Dieses Tool wird von nVidia gratis zur Verfügung gestellt. Die Zielgruppe von nVidia FX Composer sind Software-Entwicklern, welchen die Programmierung von Shadern erleichtert werden soll.

b) ATI RenderMonkey™ [ATI01]: Dieses Tool wird von ATI in Konkurrenz zum nVidia FX Composer ebenfalls gratis zur Verfügung gestellt.

c) Visual Editor [RBOR01]: Dieses Tool wurde im Rahmen einer Diplomarbeit an der Universität Paderborn entwickelt. Das Ziel des Produktes war das gleiche wie dasjenige dieser Studienarbeit. Es besitzt jedoch die folgenden Nachteile:

- Die Benutzeroberfläche ist nicht intuitiv bedienbar.
- Shader werden nur grafisch dargestellt. Eine höhere Abstraktion der Shader wäre wünschenswert.

d) rtzen RT/shader: Diese Software wurde von der Firma rtzen entwickelt und vermarktet. Es hatte dieselben Ziele wie die vorliegende Studienarbeit. Es wurde jedoch im Jahr 2005 vom Markt genommen, nachdem die Firma rtzen von einem Konkurrenten aufgekauft worden ist.

2 Durchführung der Arbeit

2.1 Involvierte Personen

An der Arbeit waren folgende Personen beteiligt:

Person	Funktion
Dr. Urs-Martin Künzi	Betreuer der Arbeit.
Christian Oberholzer	Projektausführend

2.2 Entwicklung

Die Entwicklung wurde nach den definierten Zielen und den gegebenen Risiken gegliedert und in kleinen Schritten durchgeführt. Dabei wurde darauf geachtet, dass die riskantesten Teile der Software zuerst implementiert wurden, so dass bei allfälligen Problemen noch genug Zeit für eine angemessene Reaktion zur Verfügung gestanden hätte. Die einzelnen Schritte wurden in Form von Prototypen implementiert. Es handelt sich dabei um die folgenden Prototypen:

Prototyp 1: User Interface

Der erste Prototyp umfasste einen Entwurf der Benutzeroberfläche. Das Ziel war dabei darauf zu achten, dass die Benutzeroberfläche möglichst intuitiv zu bedienen ist, und Ähnlichkeiten mit bestehender Modellierungssoftware aufweist, so dass es von der primären Zielgruppe, den Grafikern, möglichst intuitiv zu bedienen ist. Der Prototyp wurde termingerecht am 14.4.2006 fertiggestellt.

Prototyp 2: Shadergraph Plugin

Der zweite Prototyp implementiert die Umwandlung des Graphen in den Quellcode des Shaders. Diese Konvertierung ist in einem Plugin implementiert, so dass shaded erweitert werden und mehrere verschiedene Shadersprachen unterstützen könnte. Dazu definiert das Plugin die für den Graphen verwendbaren Elemente und deren Umwandlung in Quellcode. Der Prototyp wurde termingerecht innerhalb drei Wochen implementiert und am 22.5.2006 fertiggestellt.

Prototyp 3: Shadergraph Darstellung

Der dritte Prototyp implementiert die Plugin-unabhängige Darstellung und das Editieren eines Graphen von der Benutzeroberfläche aus. Der Prototyp wurde eine Woche früher als geplant am 25.5.2006 fertiggestellt.

Prototyp 4: Codedarstellung und -Export

Die im vierten Prototyp neu implementierten Komponenten sind die Codeansicht und das Exportieren von Shadern. Der Prototyp wurde termingerecht am 1.6.2006 fertiggestellt.

Prototyp 5: Vorschau-Ansicht

Der fünfte und letzte Prototyp umfasst im Vergleich zum vierten Prototypen zwei neue und grosse Komponenten; die Vorschau-Ansicht und die Szenenhierarchieansicht. Diese beiden Elemente der Benutzeroberfläche ermöglichen es dem Benutzer das Resultat der Shader in Echtzeit visuell zu sehen und deren Darstellung so zu beeinflussen, dass er eventuelle Fehler im Shader entdecken kann. Der Prototyp wurde mit einer Woche Verspätung am 23.6.2006 fertiggestellt.

Betaversion

Auf den fünften Prototypen folgte die Betaversion von shaded. Sie enthält wie der fünfte Prototyp sämtliche Anforderungen. In den letzten zwei Wochen der Arbeit wurde die Betaversion getestet und gefundene Fehler wurden behoben.

2.3 Erreichte Ziele

Die folgende Tabelle gibt einen kurzen Überblick über die erreichten MUSS-Ziele:

Nr	Kurzbeschreibung	Status
1	Laden, Speichern und Exportieren von Materialien	Erreicht
2	Modelle importieren	Erreicht
3	Laden und Speichern der aktuellen Szene	Erreicht
4	Veränderung der Szene. Dazu gehören unter Anderem das einfügen von Modellen und Materialien und das Zuweisen von Materialien zu Modellen.	Erreicht
5	Veränderungen des Shadergraphen. Dazu gehören unter Anderem das Einfügen von Elementen oder das Verbinden von Elementen.	Erreicht
6	Definition und Konvertierung von Shadergraphen. Diese Anforderung betreffen nur den Entwurf der Software, nicht aber den Benutzer.	Erreicht

Des weiteren wurden die folgenden zusätzlichen Features implementiert:

Nr	Kurzbeschreibung	Status
1	Globalisierung der Applikation	Teilweise erreicht
2	Frames: Frames sind ein neu entwickeltes Konzept im Shadergraphen.	Erreicht

2.4 Ausblick

Letztendlich als ein Prototyp (oder auch „Proof-of-Concept“) implementiert, könnte shaded in Fortsetzungsarbeiten um die folgenden Elemente erweitert werden:

- Verbesserung der Benutzeroberfläche. Die Benutzeroberfläche könnte gründlich getestet und basierend auf dem Feedback von Grafikern noch einmal überarbeitet werden.
- Verbesserung des Renderings. Dazu gehört die Erweiterung der Arbeit, so dass der Renderer als eigenständiges Konzept implementiert ist. Zusätzlich könnte der Renderer erweitert werden. Beispiele für die Möglichkeiten sind Schatten oder High Dynamic Range Rendering.
- Unterstützung von Animationen. shaded unterstützt die Entwicklung von Shadern mit Animationsanteilen nicht. Diese Feature könnte gut in einer zukünftigen Arbeit umgesetzt werden.
- Implementierung von shaded als Plugin. Eine Alternative zur Benutzeroberfläche von shaded wäre es, die Funktionalität als Plugin eines der grossen Modellierungsprogramme umzusetzen und so direkt in das Programm zu integrieren.

Teil B - Projektmanagement

Kapitel I: Projektplan

1 Einführung

Dieses Dokument beinhaltet die detaillierten Informationen zum Projektmanagement, insbesondere die Definition von Meilensteinen und Releases sowie die Risikoanalyse.

2 Projektübersicht

2.1 Ziel

Das Ziel der Studienarbeit ist es, einen Editor zu entwickeln in welchem so genannte Shader über ein grafisches Interface entworfen werden können. Genaue Zielbeschreibungen sind der Aufgabenstellung (siehe Kapitel 2) zu entnehmen.

2.2 Annahmen und Einschränkungen

Es wird von einer Arbeitszeit von 14 Stunden pro Woche ausgegangen. Sollte sich während der Entwicklung der Umfang der Software als zu gross erweisen, wird Funktionalität niedriger Priorität gestrichen.

Zusätzlich soll nur ein Resultat auf Stufe Prototyp bzw. ein „Proof of Concept" entstehen. Das bedeutet, dass die Funktionalität nicht den Rahmen eines einsetzbaren Produktes erreicht, sondern dass bloss gezeigt wird, dass diese erweiterte volle Funktionalität gut implementiert werden könnte.

2.3 Organisationsstruktur

Die Studienarbeit wird von Christian Oberholzer durchgeführt und von Dr. Urs-Martin Künzi betreut.

2.4 Termine

Die folgenden Termine sind von der HSR vorgegeben:

Datum	Beschreibung
3. April 2006	Start der Studienarbeit
7. Juli 2006	Abgabe einer Kurzbeschreibung der Arbeit
7. Juli 2006	Abgabe der Studienarbeit

Während der gesamten Projektdauer findet jeweils einmal pro Woche eine Besprechung mit dem Betreuer statt.

2.5 Prozessmodell

Als Prozessmodell wird der Rational Unified Process (RUP) verwendet.

2.6 Aufteilung der Software

Die gemäss Aufgabenstellung zu erstellende Software wird in verschiedene Komponenten aufgeteilt, so dass sie in aufeinander folgenden Arbeitsschritten entwickelt werden kann. Diese Komponenten sind:

- Shadergraph Steuerelement: Dieses Steuerelement ist das Hauptelement des zu entstehenden Editors. Wie in der Aufgabenstellung beschrieben wird dieses Steuerelement einen Shader mit

Hilfe einer Graphenstruktur graphisch darstellen. Durch das Verändern des Graphen kann der Shader verändert werden. Das Steuerelement ist jedoch nur für dessen Visualisierung zuständig.

- Shadergraph Plugin: Diese Komponente implementiert die Logik hinter dem Shadergraph Steuerelement. Jedes Shadergraph Plugin stellt die verfügbaren Graphkomponenten einer Shader-Sprachversion zu Verfügung und übernimmt die Konvertierung des Graphen in Quelltext.
- Export: Durch Shadergraph Plugins erstellte Shader sollen in beliebige Formate exportiert werden können. Ein Shader kann in verschiedene Dateiformate exportiert werden. Diese Komponente übernimmt das Exportieren eines Shaders.
- Darstellung: Die von der Exportkomponente generierte Ausgabe wird von dieser Komponente auf der Benutzeroberfläche angezeigt. Dadurch kann der Benutzer des Editors das Endprodukt seiner Arbeit sehen und durch entsprechende Veränderungen im Shadergraphen Einfluss darauf nehmen wenn er will.
- Vorschau: Diese Komponente bietet eine Vorschau des im Graphen editierten Materials. Die Vorschau wird bei Veränderungen des Graphen laufend aktualisiert und bietet so eine direkte visuelle Vorschau des Resultates. Dadurch ist es dem Grafiker möglich den Shader optimal auf sein Modell anzupassen.
- User Interface: Diese Komponente besteht aus den für eine Benutzeroberfläche nötigen Elementen.
- Applikation: Diese Komponente stellt das Bindeglied zwischen allen bisher erwähnten Komponenten dar. Sie kontrolliert das Laden des Editors und stellt die Verbindungen zwischen den einzelnen Komponenten her.

Die folgende Abbildung visualisiert die erwarteten Zusammenhänge zwischen den Komponenten.

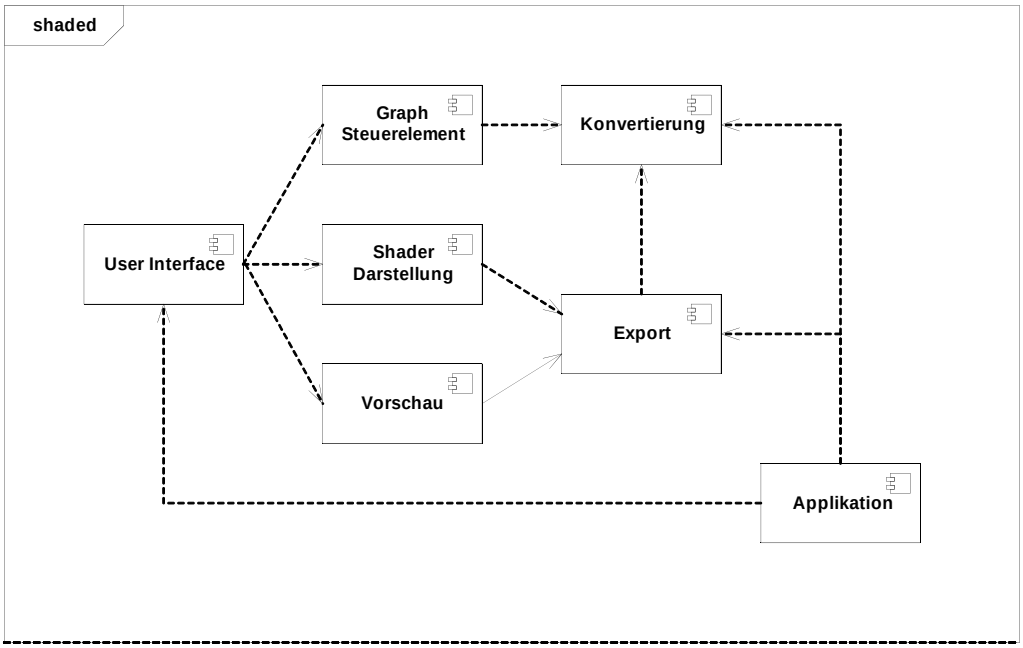


Abbildung 2: Komponenten von shaded

3 Projektphasen und Meilensteine

Für diese Studienarbeit sind nicht alle Phasen des RUP relevant. Einerseits sind die Resultate der Inception-Phase mit der vorgegebenen Aufgabenstellung bereits vorhanden und andererseits ist die Transition-Phase gemäss Aufgabenstellung nicht gefordert. Dadurch konnten die zur Verfügung stehenden 14 Wochen auf die beiden Phasen Elaboration und Construction aufgeteilt werden. Das Ende jeder Phase stellt gleichzeitig

das Erreichen eines Meilensteins dar.

Basierend auf der in Abschnitt 6 dokumentierten Risikoanalyse und der im letzten Abschnitt gezeigten erwarteten Abhängigkeiten wurden für diese Studienarbeit die Phasen und Meilensteine gemäss der nachfolgenden Tabelle festgelegt:

RUP Phase	Label und Kurzbeschreibung	Meilenstein	Datum	Zeit
Elaboration 1	Analyse: Analyse und UI Design	MS1	16.4.2006	2 Wochen
Construction 1	Shadergraph Plugin	MS2	14.5.2006	4 Wochen
Construction 2	Shadergraph Steuerelement	MS3	4.6.2006	3 Wochen
Construction 3	Export: Export und Shader Darstellung	MS4	11.6.2006	1 Woche
Construction 4	Vorschau: Vorschau-Steuerelement	MS5	25.6.2006	2 Wochen
-	Reserve: Zeitreserve und Dokumentation	MS6	7.7.2006	2 Wochen

Das Resultat jeder Phase, das heisst auch jedes erreichten Meilensteins, ist ein Prototyp der vorhergehenden Phase. Da die Funktionalität von shaded inkrementell mit jeder Phase erweitert wird, wird der Prototyp am Ende jeder Phase näher am Ziel der Aufgabenstellung sein. Es folgt eine Beschreibung der sechs Phasen:

3.1 Elaboration 1: Analyse

Während dieser Phase entstehen zuerst sämtliche Dokumente welche für die gesamte Projektdauer von Bedeutung sind. Danach wird der Analyse-Prozess durchlaufen. Dies beinhaltet das Finden der relevanten Use-Cases, darauf basierend das Festlegen der Requirements und sclhliesslich die Domain Analyse. Falls nötig wird auf aufbauend auf den Resultaten der Analyse der Projektplan angepasst.

Parallel dazu werden die vom Projekt anvisierten Interessensgruppen analysiert und es werden die Anforderungen an die Benutzeroberfläche entwickelt. Später wird unter Verwendung der Resultate der erste Prototyp entwickelt über welchen die Benutzeroberfläche getestet werden soll.

3.2 Construction 1: Shadergraph Plugin

In dieser Phase wird ein Shadergraph Plugin für HLSL mit dem Shadermodel 3.0 als Ziel implementiert. Zusätzlich soll ein Interface für alle Shadergraph Plugins entworfen werden. Die Realisierung soll in kleinen Schritten mit einem klaren Ziel geschehen. Dazu wird vor jedem dieser Schritte das bestehende Design angepasst und danach die neue Funktionalität implementiert. Es ist zu erwarten, dass dieser Teil der Software am komplexesten und schwierigsten zu entwickeln ist. Deshalb werden unter Berücksichtigung des Risikos vier Wochen dafür eingeplant.

3.3 Construction 2: Shadergraph Steuerelement

Aufbauend auf den Resultaten der vorhergehenden Phase wird in dieser Phase ein Steuerelement realisiert welches es ermöglicht den Shadergraph auf der Benutzeroberfläche darzustellen und zu editieren.

3.4 Construction 3: Export

In dieser Phase werden die zwei Komponenten Export und Darstellung entworfen, realisiert und getestet.

3.5 Construction 4: Vorschau

In dieser letzten der aufbauenden Phasen wird die Vorschau-Komponente entworfen, realisiert und getestet.

3.6 Reserve

Diese letzte „Phase“ dient als Zeitreserve und zur Verbesserung der Dokumentation. Erfahrungsgemäss

nimmt die Korrektur und das Ausformulieren der Dokumentation viel Zeit in Anspruch. Es besteht die Möglichkeit, dass ein Teil der Zeitreserve benötigt wird, um Fehlplanungen in den vorhergehenden Phasen zu korrigieren.

Zusätzlich kann in dieser Phase das fertige Produkt noch einmal getestet und wenn nötig korrigiert werden.

4 Prototypen

Jede der im letzten Abschnitt erwähnten Phasen resultiert in einem Prototypen. Die folgende Tabelle gibt einen Überblick:

RUP Phase	Meilenstein	Prototyp Nummer	Prototyp Bezeichnung
Elaboration 1	MS1	PT1	Prototyp User Interface
Construction 1	MS2	PT2	Prototyp Konvertierung
Construction 2	MS3	PT3	Prototyp Graph
Construction 3	MS4	PT4	Prototyp Export
Construction 4	MS5	PT5	Prototyp Testversion
-	MS6	PT6	Resultat

Jeder der Prototypen baut auf den vorhergehenden Prototypen auf und erweitert ihn um die in der Phase definierte Funktionalität.

5 Artefakte

Die folgende Tabelle beschreibt welche Artefakte zu welchem Meilenstein vorhanden sein müssen.

Meilenstein	Beschreibung
MS1: Analyse	Anforderungsspezifikationen.odt Aufgabenstellung.odt Domain Analyse.odt Programmierrichtlinien.odt Projektplan.odt TechBericht - User Interface.odt Zeitplanung.odt
MS2: Shadergraph Plugin	TechBericht - Shadergraph Plugin.odt
MS3: Shadergraph Steuerelement	TechBericht - Graph Steuerelement.odt
MS4: Export	
MS5: Vorschau	TechBericht - Scene.odt
MS6: Projektabgabe	Abstract.odt Dank.odt Glossar.odt kurzfass.doc Management Summary.odt Schlussbericht Projektmanagement.odt Shaderprogrammierung.odt Persönlicher Bericht.odt TechBericht - Übersicht.odt TechBericht – Kern.odt TechBericht – Umsetzung.odt TechBericht - Test.odt TechBericht - Schlussfolgerungen.odt

6 Risikoanalyse

Es ist schwierig, eine fundierte und nutzbringende Risikoanalyse bei einem kleinen Projekt wie shaded zu schreiben. Das Spezielle liegt insbesondere darin, dass bei einer Arbeit im „geschützten Umfeld“, wie eben bei einer Studienarbeit, praktisch keine Risiken mehr vorhanden sind. Statt dessen müssen allgemeingültige Risiken wie zum Beispiel das Risiko des Datenverlustes aufgeführt werden. Es existieren ebenfalls keine statistischen Informationen, auf deren Basis man die Risikoanalyse hätte aufbauen können. Statt dessen wurden deshalb eigene Erfahrungen und Schätzungen eingebracht. Die Ergebnisse der Analyse sind in der nachfolgenden Tabelle aufgezeigt:

Risiko	Auswirkung	Massnahme	Kosten der Massnahme [h]	Max. Schaden [h]	des EintretensWahrscheinlichkeit	Gewichteter Schaden [h]
RS1: Konvertierungskomponente wurde unterschätzt.	Verzögerung im Projektverlauf, Mehraufwand.	Zeitreserve in Construction 1, Zeitreserve am Projektschluss	-	28	50%	14
RS2: Vorschau-Ansicht wurde unterschätzt.	Verzögerung im Projektverlauf, Mehraufwand.	Zeitreserve in Construction 4, Zeitreserve am Projektschluss	-	14	25%	3.5
RS3: Ausfall Testsystem	Ersatz muss beschafft werden	Zweites Testsystem Zuhause	-	3	10%	0.3
RS4: Datenverlust	Teile der Arbeit gehen verloren.	Einsatz von Perforce, häufige Commits. Backups	-	5	1%	0.05
RS5: Krankheit, Unfall, etc. Ausfall	Verzögerung im Projektverlauf, ev. Projektabbruch	Keine	-	156	5%	7.8
Total			0	141		25.65

In der Zeitplanung wird der berechnete gewichtete Schaden im Umfang von 22.65 Stunden über die zwei Reservewochen am Schluss der Arbeit eingeplant.

7 Qualitätsmassnahmen

7.1 Dokumentation

Die Aktualisierung der Dokumente hat eine hohe Priorität. Zudem wird der Quellcode gemäss den Richtlinien (siehe Anhang V) gestaltet und in sinnvollem Umfang mit den Dokumentationselementen von Microsoft Visual Studio.NET 2005 versehen.

7.2 Tests

Um eine möglichst hohe Stabilität des Endproduktes zu erreichen werden ausführliche Tests gemacht. Nach jeder Phase soll der entstandene Prototyp getestet sein, so dass in den folgenden Phasen ohne Probleme darauf aufgebaut werden kann. Am Schluss soll ein Testdokument entstehen in welchem die gemachten Tests beschrieben werden.

Kapitel II: Anforderungsspezifikation

1 Vision

1.1 Problemstellung

In der üblichen Praxis werden Shader für 3-Dimensionale Modelle in einer Art IDE entwickelt. Beispiele dafür sind die folgenden Programme:

- nVidia FX Composer [NVIDIA01]
- ATI RenderMonkey™ [ATI01]

Diese Programme sind sehr gute Entwicklungswerkzeuge für Software-Entwickler und Programmierer. Sie besitzen einen Codeeditor mit Syntax-Highlighting und eine Vorschau-Ansicht, mit welcher das Resultat des editierten Shaders sofort sichtbar ist.

Es ist jedoch in Projekten eine übliche Praxis möglichst grosse Teile der Kontrolle über das Rendering von Modellen an Grafiker abzugeben, welche damit die visuelle Erscheinung des Projektes genau nach ihren Vorstellungen abstimmen können. Je weiter die Entwicklung der Grafikkarten geht, umso wichtiger werden Shader beim Rendering von Modellen. Es wäre deshalb wünschenswert, wenn Grafiker diese Programme erstellen könnten. Die Probleme dabei sind:

- Grafiker sind keine Programmierer. Sie möchten darauf verzichten, eine Programmiersprache lernen zu müssen. Daher sind die oben erwähnten Programme, welche das Programmieren eines Shaders erleichtern, für Grafiker nicht geeignet.
- Da die Grafiker gezwungen sind zu programmieren, brauchen sie die Unterstützung eines Programmierers. Dieser verliert dabei Zeit, die er in die Weiterentwicklung des Projektes investieren könnte.

Eine Software welche es ermöglicht Shader mit dem Komfort der oben erwähnten Tools zu entwickeln, jedoch das Erlernen einer Shaderprogrammiersprache nicht voraussetzt, könnte die beiden erwähnten Probleme lösen. Die Lösung dieser Probleme würde Grafikern und Programmierern eine Zeitersparnis bringen und es ermöglichen Produkte schneller und billiger herzustellen.

1.2 Interessensgruppen

Dieser Abschnitt wird die an shaded interessierten Gruppen (in Englisch: Stakeholders) identifizieren und deren primäre Ziele analysieren.

Die folgenden Interessensgruppen konnten identifiziert werden:

Grafiker

Grafiker sind dafür verantwortlich, 3-Dimensionale Modelle zu erstellen. Sie erstellen diese Modelle üblicherweise mit Hilfe einer entsprechenden Software wie zum Beispiel 3D-Studio Max oder Maya. Ihre Modelle werden dann mittels einer Grafik-Engine gerendert. Oft haben Grafik-Engines eigene, spezielle Anforderungen, welche die von den Grafikern erstellen Modelle zu erfüllen haben. Wie in der Problemstellung erwähnt, ist es dann zusätzlich die Aufgabe von Grafikern, Shader zu entwickeln, so dass ihre Modelle optimal dargestellt werden. Grafiker könnten, in Bezug auf shaded, folgende Interessen haben:

Die Grafiker möchten ihre Vorstellung der visuellen Erscheinung eines Modelles auf die Anwendung übertragen können. Dies geschieht über Shader, welche der Grafiker in seiner gewohnten Arbeitsumgebung mit seinen gewohnten Werkzeugen und ohne Programmierkenntnisse erstellen möchte.

Software Entwickler

Die Software Entwickler sind dafür verantwortlich parallel zu den Grafikern die Software eines Projektes (normalerweise eine Simulation oder ein Spiel) zu entwickeln. Dies umfasst einerseits die Grafik-Engine, welche es ermöglicht die erstellten 3-Dimensionalen Modelle im Kontext der Applikation anzuzeigen, andererseits gehören dazu aber auch viele andere Elemente wie Simulationselemente, Physik-Engines, Spiellogik und Ähnliches. Ihre Interessen sehen folgendermassen aus:

Die Software Entwickler sind nur daran interessiert wie Modelle gerendert werden. Sie möchten die gestalterischen Anteile der Arbeit soweit als möglich an die Grafiker abgeben können, so dass diese entscheiden können wie etwas gerendert wird. Daher möchten sie, dass die Grafiker Shader selber entwickeln können. Sie sind ebenfalls daran interessiert, dass die Grafiker möglichst selbständig arbeiten können und keine Unterstützung durch Programmierer benötigen, so dass sich die Entwickler auf ihre Aufgaben konzentrieren können.

Hobbygrafiker und Hobbyprogrammierer

Hobbygrafiker und Hobbyprogrammierer sind diejenigen Leute welche neben der alltäglichen Arbeit als Hobby grafische Anwendungen erstellen. Sie organisieren sich in kleinen Gruppen und versuchen in ihrer Freizeit mit möglichst wenig Aufwand ihre oft kleineren Projekte zu implementieren.

Zusätzlich zu den erwähnten Interessen der Grafiker, respektive der Programmierer, sind sie daran interessiert, dass sich Input- und Output-Daten der von ihnen verwendeten Tools an offene Standards halten, so dass diese durch weitere gut verfügbare Software bereitgestellt, bzw. verarbeitet werden können.

Manager und Firmen

Diese Interessensgruppe umfasst alle Leute welche für die Leitung von Projekten verantwortlich sind. Sie sind daran interessiert, dass Programmierer und Grafiker die gestellten Aufgaben möglichst effizient und schnell beenden können.

1.3 Zusammenfassung der Bedürfnisse

Zusammengefasst können folgende Bedürfnisse identifiziert werden:

- Shader sollen visuell und ohne Programmierkenntnisse erstellt werden können.
- Programmierer eines Projektes sollen Einfluss nehmen können auf die Dateiformate der verwendeten Input- und Output-Daten.

2 Aktoren

Die folgenden Aktoren konnten analysiert werden:

Name	Beschreibung
Benutzer	Der Benutzer ist der primäre Aktor des Systems. Er erstlelt mit shaded seine Shader.
Modellimport Plugin	Import Plugins sind externe Komponenten welche zu shaded hinzugefügt werden können. Sie werden von shaded verwendet, um Modelle eines bestimmten Dateiformates zu laden.
Shadergraph Plugin	Shadergraph Plugins sind externe Komponenten, welche zu shaded hinzugefügt werden können. Sie werden von shaded dazu verwendet, um einen Shadergraphen zu definieren und diesen später in einen Shader zu konvertieren.
Export Plugin	Export Plugins sind externe Komponenten welche zu shaded hinzugefügt werden können. Sie werden von shaded dazu verwendet, Shader in einem bestimmten Dateiformat abzuspeichern.

3 Use-Case Modell

3.1 Übersicht

Für eine bessere Übersicht in den Use-Case Diagrammen wurden die Use-Cases in die nachfolgenden drei Gruppen aufgeteilt. Eine Beschreibung der gezeigten Use-Cases im „Brief"-Format kann im nächsten Abschnitt gefunden werden. Auf eine Beschreibung im „Fully Dressed"-Format wurde nach Absprache mit dem Betreuer aus Zeitgründen verzichtet.

Daten speichern

Diese Use-Cases beschreiben wie der Benutzer die verschiedenen Daten laden, speichern, importieren und exportieren kann. Es sind die die in der folgenden Grafik gezeigten Use-Cases definiert.

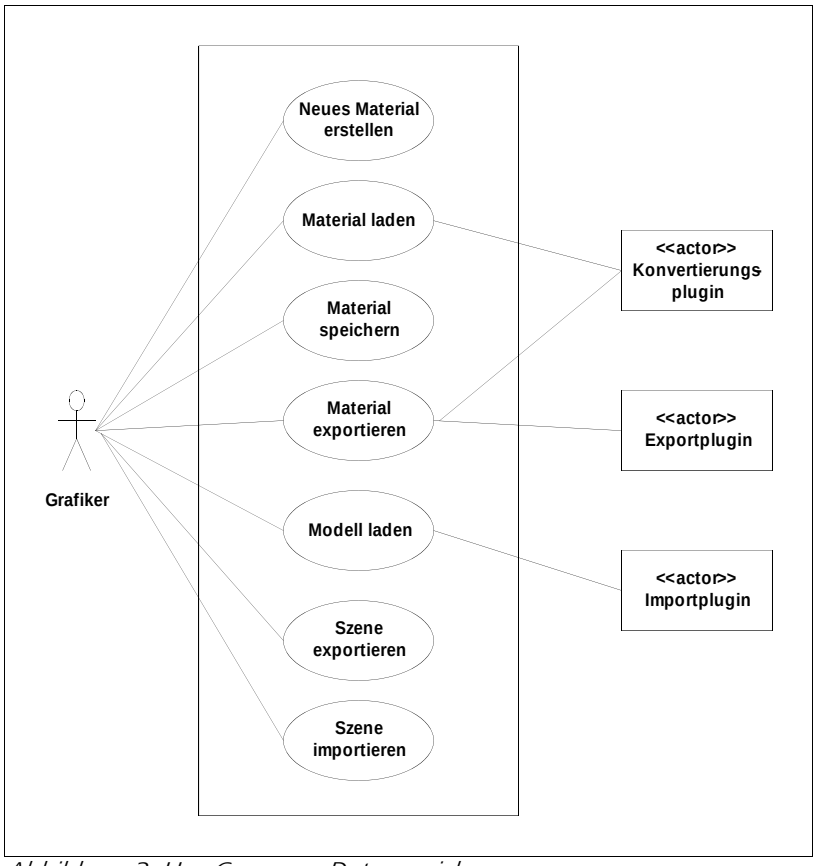


Abbildung 3: Use-Cases zur Datenspeicherung

Szene ändern

Diese Use-Cases beschreiben wie der Benutzer die Anordnung der Modelle in der aktuellen Szene und die Ansicht darauf verändern kann. Es sind die die in der folgenden Grafik gezeigten Use-Cases definiert.

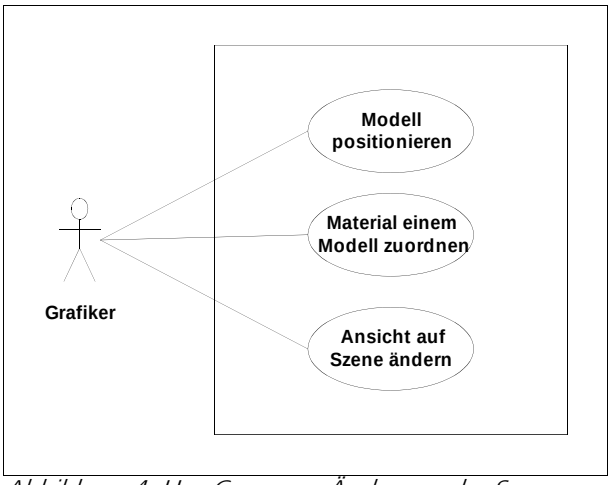


Abbildung 4: Use-Cases zur Änderung der Szene

Shader-Graph ändern

Diese Use-Cases beschreiben wie der Benutzer den Shader-Graph und damit einen Shader verändern kann. Es sind die die in der folgenden Grafik gezeigten Use-Cases definiert.

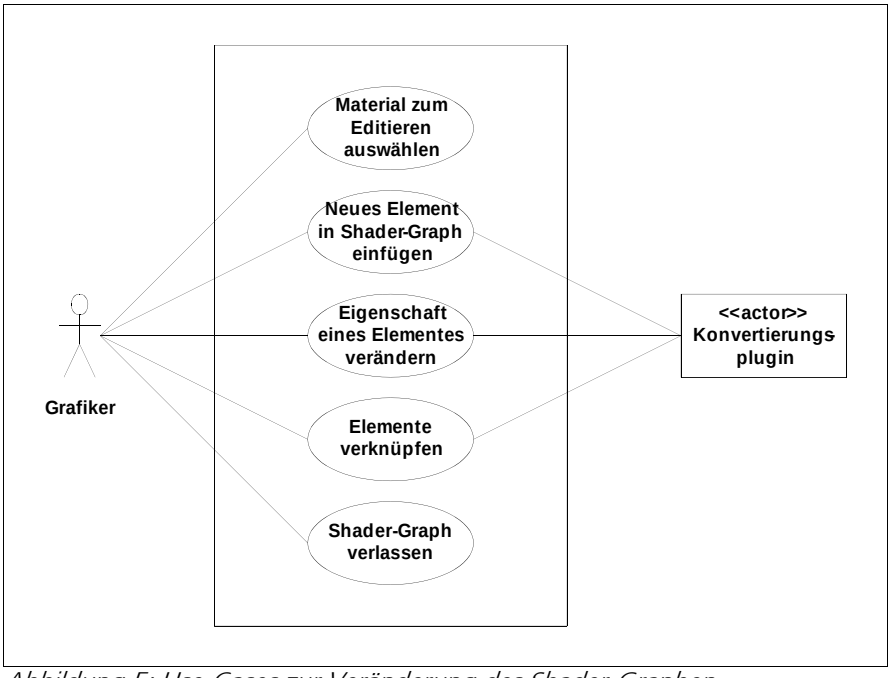


Abbildung 5: Use-Cases zur Veränderung des Shader-Graphen

3.2 Use Cases

Im folgenden werden alle Use Cases im „Brief"-Format beschrieben.

UC1: Neues Material erstellen

Der Benutzer hat das Programm geöffnet, Modelle importiert und möchte nun ein neues Material und damit

einen neues Shader erstellen. Er wählt dazu die entsprechende Option von shaded, worauf das Programm ein neues Material erstellt und dem Benutzer anzeigt.

UC2: Material laden

Falls der Benutzer bereits einmal ein Material erstellt und gespeichert hat, möchte er vielleicht das Material laden, um entweder daran weiterzuarbeiten oder es zu verändern. Dazu wählt er die entsprechende Option von shaded aus. Darauf zeigt ihm die Applikation eine Auswahl über alle Materialien, welche auf dem Computer vorhanden sind. Der Benutzer kann nun das gewünschte Material wählen und shaded lädt es und zeigt es an.

UC3: Material speichern

Nachdem ein Benutzer ein Material erstellt und dessen Shadergraph bearbeitet hat, möchte er das Material speichern, um später daran weiterzuarbeiten. Er wählt dazu die entsprechende Option von shaded aus. Die Applikation präsentiert ihm eine Möglichkeit den Ort zu wählen wo er das Material speichern möchte, und speichert es dann mit dem gewählten Namen am ausgewählten Ort.

UC4: Material exportieren

Sobald der Benutzer mit dem Editieren eines Materials fertig ist oder sehen möchte, wie das Material im Einsatz im echten Projekt aussieht, exportiert er das Material. Er wählt dazu die entsprechende Option von shaded aus. Die Applikation präsentiert ihm eine Möglichkeit Speicherort, Namen und Format zu wählen, mit welchen er das Material exportieren möchte. shaded wählt dann das mit dem gewählten Format assoziierte Material-Exportplugin und gibt diesem eine Möglichkeit den Benutzer nach zufälligen Informationen zu fragen. Nachdem das Plugin dies getan hat, übergibt shaded dem Plugin Ort und Namen unter welchen der Benutzer das Material exportieren möchte und das Plugin exportiert das Material dorthin.

UC5: Material entfernen

Wenn der Benutzer ein gewisses Material in der aktuellen Szene nicht mehr benutzen möchte, so kann er es entfernen. Dazu selektiert er zuerst das Material und wählt danach die entsprechende Option von shaded zum Entfernen des Materials. Die Applikation löscht darauf zuerst eventuell noch vorhandene Assoziationen zwischen dem Material und einem oder mehreren Modellelementen. Danach entfernt shaded das Material aus der aktuellen Szene.

UC6: Modell importieren

Der Benutzer möchte für ein bestimmtes Modell ein neues Material erstellen. Dazu muss er das Modell zuerst laden. Er wählt die entsprechende Option von shaded. Die Applikation gibt ihm eine Auswahl über alle Modelle welche auf dem Computer vorhanden sind. Nachdem der Benutzer das gewünschte Modell ausgewählt hat evaluiert shaded den Typ des Modells und sucht das mit diesem Typ assoziierte Modell-Importplugin. Danach übergibt shaded die Kontrolle an das Plugin um das Modell zu laden, worauf es das Modell und alle dazugehörenden Modellelemente lädt.

UC7: Standardgeometrie erstellen

Dieser Use-Case tritt ein, wenn der Benutzer ein Material erstellen möchte, dieses aber nicht auf eines seiner eigenen Modelle anwenden, sondern an Standard-Geometrie testen möchte. Dazu wählt er die entsprechende Option ,um eines der folgenden Standard-Modelle zu erstellen:

- Zylinder
- Quader
- Ebene

shaded erstellt daraufhin ein neues Modell welches die gewünschte Geometrie enthält und aktualisiert die Szene, damit die Änderung sofort sichtbar wird.

UC8: Szene laden

Der Benutzer hat bereits einmal eine Szene gespeichert und möchte diese nun wieder laden. Er wählt dazu die entsprechende Option von shaded. Die Applikation gibt ihm eine Auswahl über alle Szenen, welche auf dem Computer vorhanden sind. Nachdem der Benutzer die gewünschte Szene gewählt hat, lädt shaded die Szene und alle damit assoziierten Materialien, Modelle und Lichtquellen.

UC9: Szene speichern

Nachdem der Benutzer einige Modelle geladen, diese im Raum verschoben oder rotiert hat, ihnen Materialien zugeordnet hat und Lichtquellen definiert hat, möchte er die aktuelle Anordnung für eine spätere Arbeitssitzung speichern. Er wählt dazu die entsprechende Option von shaded. Die Applikation gibt ihm die Möglichkeit, Ort und Namen der Szene zu wählen und speichert sie dann mit dem gewählten Namen am gewählten Ort.

UC10: Szene zurücksetzen

Wenn der Benutzer mit dem Editieren der Materialien in der Szene fertig ist und eine neue Szene beginnen möchte, so kann er eine entsprechende Option in shaded wählen. Die Applikation entfernt darauf alle Materialien, Modelle und Lichtquellen aus der Szene und setzt damit die Benutzeroberfläche auf denselben leeren Zustand zurück, wie sie nach Programmstart ist.

UC11: Modell positionieren

Nachdem der Benutzer verschiedene Modelle importiert hat, möchte er sie in eine korrekte Relation zueinander setzen. Er wählt dazu zuerst das Modell, dessen Position oder Orientierung er ändern möchte. Danach wählt er entweder die Option um ein Modell zu rotieren oder die Option um ein Modell zu verschieben. Zuletzt kann er mit dem gewählten Werkzeug die Position oder Orientierung des gewählten Modells direkt in der Vorschau-Ansicht von shaded verändern. shaded aktualisiert dazu die Vorschau laufend, so dass er das Resultat seiner Änderungen direkt sieht.

UC12: Modell entfernen

Dieser Use-Case tritt ein, wenn der Benutzer ein importiertes Modell aus der aktuellen Szene entfernen möchte. Dazu wählt er das Modell welches er entfernen möchte, und entfernt es über die entsprechende Option von shaded. Die Applikation entfernt das Modell und alle dazu gehörenden Modellelemente. Danach aktualisiert shaded die Vorschau, so dass der Benutzer das Resultat der Änderung sofort sehen kann.

UC13: Ansicht auf Szene ändern

In der Vorschau-Ansicht von shaded kann die aktuelle Szene unter einem bestimmten Blickwinkel gesehen werden. In der Fachsprache wird von einer Kamera geredet und angenommen, dass der Benutzer die Szene so sieht als würde er das Life-Bild ansehen, das die Kamera von der Szene aufnimmt. Falls die Kamera zur Ansicht auf das editierte Material oder Modell nicht optimal positioniert ist, kann der Benutzer deren Ausrichtung, Position oder Zoomfaktor ändern. Der Benutzer wählt dazu aus, ob er den Blickwinkel um die Szene rotieren möchte, ob er die Szene vergrössern oder verkleinern möchte oder ob er den Blickwinkel relativ zur Szene verschieben möchte. Nachdem er gewählt hat was er verändern möchte, kann er die Ansicht auf die Szene direkt in der Vorschau-Ansicht von shaded verändern. shaded aktualisiert dazu die Vorschau laufend, so dass er das Resultat seiner Änderungen direkt sehen kann.

UC14: Material einem Modellelement zuordnen

Nachdem der Benutzer ein Modell importiert und entweder ein neues Material erstellt oder eines geladen hat möchte er das Material einem Teil eines Modells zuordnen. Er kann dazu mit shaded das Material auswählen und dem Modellelement direkt zuordnen. Danach zeigt shaded direkt in der Vorschau-Ansicht wie das Modell, respektive dessen Modellelement, mit dem neuen Material aussehen wird.

UC15: Lichtquelle einfügen

Wenn der Benutzer mit der Beleuchtung der Szene nicht zufrieden ist, kann er eine neue Lichtquelle einfügen. Dazu wählt er die entsprechende Option von shaded worauf die Applikation die neue Lichtquelle einer standardmässig bestimmten Stelle einfügt. Für diese Studienarbeit ist es nur möglich, Punktlichtquellen einzufügen. Es wäre aber im Interesse der Grafiker noch weitere Lichttypen zu behandeln. Nachdem die Lichtquelle eingefügt worden ist, aktualisiert shaded die Vorschau, so dass der Benutzer das Resultat der neuen Lichtquelle sofort sehen kann.

UC16: Parameter einer Lichtquelle ändern

Dieser Use-Case tritt ein, wenn der Benutzer mit den Parametern einer Lichtquelle, zum Beispiel deren Position, nicht zufrieden ist. Dazu wählt er die zu verändernde Lichtquuelle an und kann den gewünschten Parameter der Lichtquelle direkt über Elemente der Benutzeroberfläche verändern. Nachdem der Parameter verändert worden ist, aktualisiert shaded die Vorschau, so dass der Benutzer das Resultat der Veränderung sofort sehen kann.

UC17: Lichtquelle entfernen

Wenn der Benutzer der Meinung ist, dass eine Lichtquelle nicht mehr benötigt wird kann er diese entfernen. Er wählt dazu die Lichtquelle aus und danach die entsprechende Option von shaded. Die Applikation löscht darauf die Lichtquelle und aktualisiert die Vorschau, so dass der Benutzer das Resultat ohne die Lichtquelle sofort sehen kann.

UC18: Material zum Editieren auswählen

Der Benutzer hat ein neues Material erstellt oder möchte an einem bestehenden Material weiterarbeiten. Dazu wählt er zuerst das gewünschte Material aus. Danach kann er die Option wählen das Material zum editieren. Macht er das, so zeigt ihm shaded sofort den Shadergraph des Materials an.

UC19: Neues Element in Shader-Graph einfügen

Wenn der Benutzer ein Material zum Editieren ausgewählt hat und dessen Shadergraph vor sich hat, kann er ein neues Element in den Shadergraph einfügen. Wenn er dies möchte, kann er die entsprechende Option von shaded auswählen. Die Applikation zeigt ihm dann eine Auswahl der verschiedenen Elemente die er in den Graphen einfügen kann. Der Benutzer wählt das gewünschte Element und shaded fügt das Element in den Graphen ein.

UC20: Eigenschaft eines Elementes verändern

Wenn der Benutzer ein Material zum Editieren ausgewählt hat und dessen Shadergraph vor sich hat, ist es möglich, dass Elemente veränderbare Eigenschaften besitzen. Wenn er eine bestimmte Eigenschaft eines solchen Elementes ändern möchte, wählt er das Element und die Eigenschaft an. Danach kann er unter den Optionen von shaded diejenige zur Veränderung des Elementes auswählen. Die Applikation bietet ihm darauf abhängig vom Typ der Eigenschaft die Möglichkeit diese Eigenschaft zu verändern. Nachdem der Benutzer die Eigenschaft geändert hat, aktualisiert shaded den aus dem Graphen resultierenden Shader und die Vorschau-Ansicht, so dass die Änderung sofort sichtbar wird.

UC21: Elemente verbinden

Der Benutzer kann diese Aktion unternehmen, nachdem er ein Material zum Editieren ausgewählt und dessen Shadergraph vor sich hat. Er möchte nun entweder eine neue Verbindung zwischen zwei Elementen herstellen oder die Verbindung zwischen zwei Elementen ändern. shaded bietet ihm dazu direkt im Shadergraph-Steuerelement die Möglichkeit. Nachdem der Benutzer eine neue Verbindung hergestellt oder eine bestehende Verbindung geändert hat, aktualisiert shaded den aus dem Graphen resultierenden Shader und die Vorschau-Ansicht, so dass die Änderung sofort sichtbar wird.

UC22: Element löschen

Wenn der Benutzer ein Material zum editieren ausgewählt und dessen Shadergraph vor sich hat, möchte er eventuell ein Element aus dem Graphen entfernen. Dazu selektiert er das Element und wählt die entsprechende Option von shaded zum Löschen des Elementes. Die Applikation entfernt darauf zuerst eventuell noch vorhandene Verbindung zwischen dem zu löschenden Element und anderen Elementen. Danach löscht shaded das Element und aktualisiert den aus dem Graphen resultierenden Shader und die Vorschau-Ansicht, so dass die Änderung sofort sichtbar wird.

UC23: Shader-Graph verlassen

Nachdem der Benutzer zufrieden ist mit den Änderungen die er an einem Shadergraphen gemacht hat kann er den Shadergraph über eine Option von shaded verlassen, damit er später, wenn er möchte, den Shadergraphen eines anderen Materials anzeigen kann.

4 Annahmen und Abhängigkeiten

4.1 Annahmen

Es bestehen keine Annahmen.

4.2 Abhängigkeiten

shaded hat Abhängigkeiten zu folgenden Software-Bibliotheken:

- DirectX9 SDK: shaded basiert auf der „Februar 2006“-Ausgabe des DirectX 9 Software Development Kits. Dieses Development-Kit oder eine ebenfalls kompatible neuere Version kann unter http://msdn.microsoft.com/directx heruntergeladen werden.
- Die Unit Tests von shaded basieren auf der Nunit-Library der Version 2.2. Die Library kann unter http://www.nunit.org/ heruntergeladen werden.

5 Anforderungen

Im Nachfolgenden werden die genauen Anforderungen an die Software aufgeführt. Die funktionalen Anforderungen wurden dabei direkt aus den Use-Case Modellen abgeleitet. Die darauf folgenden Anforderungen zu Verwendbarkeit (Usability), Zuverlässigkeit, Geschwindigkeit (Performance) und Wartbarkeit stellen dagegen weitere nicht aus den Use-Case ableitbare Anforderungen dar.

Anforderungen der Priorität 1 müssen zwingend erfüllt sein damit die übergeordnete Anforderung als erfüllt gilt. Grundsätzlich werden die Anforderungen gemäss der zugeordneten Priorität realisiert. Es gilt dabei, dass die Priorität umso höher ist je niedriger die zugeordnete Nummer ist. Prioritäten gleicher Stufe können in zeitlich beliebiger Reihenfolge realisiert werden.

5.1 Funktionale Anforderungen

Die folgenden funktionalen Anforderungen wurden aus den Use-Cases abgeleitet:

FA1: Material serialisieren

Teilanforderung	Beschreibung	Use-Case	Priorität
FA1a: Neues Material erstellen	Der Benutzer kann innerhalb von shaded ein neues Material erstellen.	UC1	1
FA1b: Material entfernen	Der Benutzer kann ein Material aus der aktuellen Szene entfernen.	UC5	1
FA1c: Material laden	Der Benutzer kann ein Material aus einer bestehenden Materialdatei laden.	UC2	1
FA1d: Material speichern	Der Benutzer kann ein Material in eine Datei speichern.	UC3	1
FA1e: Material exportieren	Der Benutzer kann die Shader eines fertig editierten Materials in eines der vorhandenen externen Dateiformate exportieren. Für das Exportieren des Materials wird ein Interface für Exportplugins definiert. Es wird mindestens ein Plugin für DirectX9 Effect Files implementiert.	UC4	1

FA2: Modell importieren

Teilanforderung	Beschreibung	Use-Case	Priorität
FA2a: Modell importieren	Der Benutzer kann ein 3-Dimensionales Modell importieren. Dazu wird ein Interface für ein Importplugin definiert. Es wird mindestens ein Plugin für .x-Dateien implementiert.	UC6	1
FA2b: Modell entfernen	Der Benutzer kann ein 3-Dimensionales Modell und all seine Elemente aus der Szene entfernen.	UC12	1

FA3: Szene serialisieren

Teilanforderung	Beschreibung	Use-Case	Priorität
FA3a: Szene laden	Der Benutzer kann eine bestehende Szenendatei laden.	UC8	1
FA3b: Szene speichern	Der Benutzer kann die momentan editierte Szene in eine Datei zur späteren Wiederverwendung speichern.	UC9	1

FA4: Szene verändern

Teilanforderung	Beschreibung	Use-Case	Priorität
FA4a: Modell rotieren	Der Benutzer kann ein Modell in der Vorschau-Ansicht um ein oder mehrere Achsen rotieren.	UC11	2
FA4b: Modell verschieben	Der Benutzer kann ein Modell in der Vorschau-Ansicht im Raum verschieben	UC11	2
FA4c: Ansicht um Szene rotieren	Der Benutzer kann die Ansichtsperspektive in der Vorschau-Ansicht um die Szene rotieren	UC13	1
FA4d: Ansicht in Szene verschieben	Der Benutzer kann die Ansichtsperspektive innerhalb der Szene verschieben	UC13	1
FA4e: Ansicht an Szene heran oder davon weg zoomen	Der Benutzer kann die Ansicht näher an die Ansicht heran oder davon weg zoomen.	UC13	1
FA4f: Der Benutzer kann Materialien Modellen zuordnen	Der Benutzer kann ein Material einem oder mehreren Modellen zuordnen. Er kann die Zuordnung ändern.	UC14	1
FA4g: Lichtquelle einfügen	Der Benutzer kann eine neue Lichtquelle in die Szene einfügen.	UC15	1
FA4h: Lichtquelle verändern	Der Benutzer kann die Parameter (z.b. Position, Orientierung, etc.) einer Lichtqulle verändern.	UC16	2
FA4i: Lichtqulle entfernen	Der Benutzer kann eine Lichtqulle aus der Szene entfernen.	UC17	2

Teilanforderung	Beschreibung	Use-Case	Priorität
FA4j: Szene zurücksetzen	Der Benutzer kann die momentane Szene auf den leeren Ursprungszustand zurücksetzen. Geladene Materialien und Modelle werden aus dem Hauptspeicher entfernt.	UC10	1

FA5: Material/Shadergraph ändern

Teilanforderung	Beschreibung	Use-Case	Priorität
FA5a: Material zum Editieren wählen	Der Benutzer kann ein beliebiges geladenes Material zum Editieren auswählen. Es wird darauf der Shadergraph des Materials angezeigt.	UC18	1
FA5b: Element in Shadergraph einfügen	Der Benutzer kann ein neues Element eines gewählten Typs in den Shadergraph einfügen.	UC19	1
FA5c: Eigenschaften der Elemente verändern	Der Benutzer kann die Eigenschaften der Elemente des Shadergraphen verändern.	UC20	1
FA5d: Elemente verknüpfen	Der Benutzer kann die Elemente im Shadergraph miteinander verknüpfen. Jedes Element besitzt einen oder mehrere Input- und Outputkanäle. Es kann jeweils immer ein Input mit beliebig vielen Output-Kanälen verbunden werden.	UC21	1
FA5e: Elemente löschen	Der Benutzer kann nicht mehr benötigte Elemente des Shadergraphen aus dem Graphen entfernen.	UC22	1
FA5f: Shadergraph verlassen	Der Benutzer kann den Shadergraph eines Materials verlassen.	UC23	1

FA6: Shadergraph definieren und konvertieren

Teilanforderung	Beschreibung	Use-Case	Priorität
FA6a: Shadergraph Plugin Interface	Es wird das Interface für ein Konvertierungsplugin erstellt. Das Konvertierungsplugin kann definieren was für Elemente im Shadergraph zugelassen sind. Es bestimmt ebenfalls die Eigenschaften dieser Elemente. Hauptsächlich konvertiert das Konvertierungsplugin jedoch den definierten Shadergraphen in einen Vertex- und einen Pixelshader der vom Konvertierungsplugin verwendeten Shadersprache.	-	1
FA6b: Shadergraph Plugin	Es wird ein Konvertierungsplugin für DirectX9 Vertex- und Pixelshader Version 3.0 erstellt.	-	1

5.2 Verwendbarkeit

Die Bedienung von shaded wird über eine grafische Benutzeroberfläche gesteuert. Sie soll sich an der gewohnte Arbeitsumgebung der wichtigsten Benutzer orientieren und für diese dadurch einfach und intuitiv zu bedienen sein. Die einzelnen Funktionen werden nach deren Bedürfnissen so in die Benutzeroberfläche integriert, dass sie entweder sofort ersichtlich oder intuitiv auffindbar sind.

5.3 Zuverlässigkeit

shaded soll einigermassen stabil funktionieren. Das heisst das zu implementierende Shadergraph Plugin mit „Proof of Concept"-Funktionalität soll am Ende der Studienarbeit keine bekannten Fehler mehr enthalten, die einen direkten Absturz der Applikation zur Folge haben.

5.4 Geschwindigkeit

Wichtig und unmittelbar wahrnehmbar für die Benutzer von shaded ist, dass die Benutzeroberfläche auf die verschiedenen Interaktionsmöglichkeiten akzeptable Antwortzeiten liefern kann. Diese lassen sich in folgende Kategorien aufteilen:

- Wahrnehmungen: Es gibt Aktionen bei denen der Benutzer eine unmittelbare Reaktion des Systems erwartet. In shaded gehört dazu die Interaktion mit dem Shadergraphen und der

Vorschau-Ansicht. Es gehören dazu ebenfalls alle Elemente und Optionsfelder welche sich direkt auf der Benutzeroberfläche befinden. Es wird vom Benutzer erwartet, dass die Applikation auf die Interaktion mit diesen Elementen innerhalb von 100ms antwortet.

- Unmittelbare Tätigkeit: Bei vielen Aktionen erwartet der Benutzer eine schnelle Reaktion, allerdings sind Zeiten bis zu einer Sekunde durchaus noch akzeptabel. In shaded gehören dazu das Laden und Importieren von Daten (wie Materialien, Modellen und/oder einer Szene). Bei diesen Vorgängen ist eine Verzögerung von bis zu einer Sekunde akzeptabel.

Die oben erwähnten Reaktionszeiten von Prozessen innerhalb der Applikation, sollen eingehalten werden. Kann ein Prozess nicht innerhalb dieser gegebenen Zeitspanne durchgeführt werden, so muss eine entsprechende Übergangslösung implementiert werden. Dazu gehört zum Beispiel die Anzeige einer Fortschrittsleiste oder Ähnlichem, wenn der Prozess voraussichtlich viel Zeit beansprucht.

5.5 Wartbarkeit

Coding Style

Der Programmcode soll nach festgelegten Richtlinien erstellt werden. (siehe Anhang V)

Erweiterbarkeit

Die Applikation soll durch die Implementierung einer Plugin-Infrastruktur und durch die in den funktionalen Anforderungen geforderten drei Plugin-Typen erweiterbar sein. Die erwähnten drei Plugin-Typen sind:

- Modellimport Plugins
- Export Plugins
- Shadergraph Plugins

Kapitel III: Schlussbericht Projektmanagement

1 Planung

Die im Projektplan festgelegte Planung konnte ziemlich genau eingehalten werden.

Die erste der beiden Änderungen wurde nach der ersten Iteration vorgenommen und betraf den unmittelbaren weiteren Verlauf der Arbeit. Aus zwei Gründen wurde beschlossen das Shadergraph-Plugin bereits in der zweiten Iteration zu entwickeln und das Shadergraph-Steuerelement auf die dritte Iteration zu verschieben. Die Gründe sind:

e) Die Risikoanalyse hatte gezeigt, dass das Shadergraph-Plugin eines der grössten Risiken des gesamten Projektes war. Des Weiteren empfiehlt die Literatur ([CLAR01]) die riskantesten Teile eines Projektes als erstes zu implementieren, so dass die Risiken eliminiert werden können, solange noch genug Zeit übrig ist um auf eventuelle Probleme zu reagieren.

f) Mit der frühen Implementierung des Shadergraph-Steuerelementes hätte einerseits das Interface zum Plugin bereits festgelegt und andererseits ein Stub für das Plugin implementiert werden müssen. Diese beiden Varianten hätten jeweils Einschränkungen oder einen Mehraufwand bedeutet. Im Fall des Interfaces wäre die Entwicklung des Shadergraph-Plugins eingeschränkt worden. Alternativ dazu hätte auch nach Bedarf das interface geändert werden können, wodurch aber ein Mehraufwand entstanden worden wäre.

Die zweite Änderung wurde dadurch verursacht, dass die für das Projekt benötigte Grafikkarte von der Hochschule sehr lange nicht zur Verfügung gestellt wurde. Die Folge davon war, dass die aufwändigen Shader des Shadergraph-Plugins nicht getestet werden konnten. Zur Lösung des Problems wurde eine der vier für das Shadergraph-Plugin geplanten Wochen auf später verschoben. Dazwischen wurde die Entwicklung des Shadergraph-Steuerelementes eingeschoben. Dadurch konnte genug Zeit gewonnen werden bis die Grafikkarte zur Verfügung stand.

Der Rest des Projektes konnte wie geplant durchgeführt werden.

2 Aufwandsschätzungen

Den Unterschied zwischen den für Arbeiten geplanten Aufwänden und den tatsächlich benötigten Zeiten war sehr schwierig zu erfassen. Die erfassten Zeitaufwände weichen daher sehr wahrscheinlich von den tatsächlich benötigten Aufwänden ab, was es schwierig macht, gute Schlüsse zu ziehen. Die folgenden Unterschiede konnten ausgemacht werden:

- Für die Implementierung des Shadergraph-Steuerelementes wurden statt den geplanten drei Wochen nur zwei Wochen benötigt.
- Die Implementierung der Vorschau-Ansicht hingegen dauerte eine Woche länger als geplant, nämlich fünf statt vier Wochen.
- Die Dokumentation nahm trotz eingeplanten Reserven einige Zeit mehr in Anspruch als geplant.

Im Total wurden 261.75 Stunden geplant und tatsächlich 271.61 Stunden benötigt.

Teil C - Technischer Bericht

Kapitel I: Shaderprogrammierung

1 Übersicht

Dieses Kapitel soll eine kurze Einführung in die 3D- und Shaderprogrammierung geben. Damit können die Funktionsweise und Grundbegriffevon DirectX, der für diese Studienarbeit verwendeten Rendering-API, klargestellt werden. Die Einführung setzt jedoch bereits eine gewisse Grundkenntnis im Bereich 3D-Rendering voraus.

Nach einer kurzen Erklärung woher der Begriff „Shader" stammt befasst sich der erste Teil mit der Rendering Pipeline im Allgemeinen. Der zweite Teil geht näher auf die in die in der Rendering Pipeline erwähnten Vertex- und Pixelshader ein. Der letzte Teil schlussendlich zeigt die Shader-Hochsprachen welche die Programmierung von Vertex- und Pixelshadern entscheidend erleichtern können.

2 Ursprung

Der Begriff „Shader" existiert seit 1988.

In diesem Jahr führte Pixar den neuen Standard RISpec („The RenderMan Interface Specification") ein. Dieser Standard ist eine technische Spezifikationfür die Kommunikation zwischen Modellierungsprogramm und dem Renderer mit welchem fotorealistische Bilder erzeugt werden können. Die Idee dahinter ist ähnlich derjenigen hinter PostScript mit dem Unterschied, dass nicht der Aufbau von druckbaren Seiten, sondern derjenige von 3-Dimensionalen Szenen beschrieben wird. Durch diesen Standard wurde eine effektive Entkopplung von Modellierungsprogramm und Renderer erreicht. Das Modellierungsprogramm kann durch den Standard seine Daten an die Rendering-Software schicken, ohne sich darum kümmern zu müssen wie dieser intern funktioniert. Eines der Ziele der RISpec war es, dass die Spezifikation auf viele Jahre in die Zukunft hinaus aktuell bleiben sollte. Um dieses Ziel zu erreichen wurde eine sogenannte „shading language" (SL) in den Standard integriert. Diese SL war eine Neuerung und einer der grossen Unterschiede zwischen der RISpec und anderen vergleichbaren Standards dieser Zeit. Die SL erlaubte es Materialien und Oberflächen nicht nur über einen Satz von vorgegebenen Parametern zu bestimmen, sondern sie konnten durch kleine Programme, in einer C-Ähnlichen Programmiersprache, beliebig geformt und eingefärbt werden. Diese mit der SL entwickelten Programme wurden danach ebenfalls als „Prozedurale Texturen" oder eben „Shader" bekannt. (Informationen aus [WIKI05])

Pixar konnte mit der RenderMan-Technologie erstmals im Jahr 1996 durch den Film „Toy Story" grosse Erfolge feiern. Später folgten von Pixar weitere auf RenderMan basierende Filme. Die meisten hatten ebenfalls grossen Erfolg im Kino. Heute wird RenderMan oftmals in Filmen für Spezialeffekte verwendet. Beispiele dafür sind „Star Wars" und „The Lord of the Rings".

In ungefähr der gleichen Zeit, also 1996, kamen erstmals Grafikkarten auf den Markt, mit welchen das Rendern von 3-Dimensionalen Modelle über in die Hardware integrierte Algorithmen beschleunigt wurde. In den folgenden Jahren und bis heute entwickelten sich diese Grafikkarten rapide und es wurden diverse Technologien aus der Filmindustrie übernommen. Um das Jahr 2000 wurde erstmals die Technologie der SL von RenderMan kopiert und ein einer Abwandlung in die Spezifikation von DirectX 8 übernommen. Damals konnten die Grafikkarten jedoch nur sehr kleine Shader-Programme ausführen. Im Laufe der Zeit entwickelte sich die Technologie rapide und heute erlauben moderne Grafikkarten die Ausführung von sehr komplexen Shadern.

3 Die Rendering Pipeline

Normalerweise wird das Rendering von Modellen in einer Rendering Pipeline organisiert. Das heisst, dass der Grafikkarte Modelldaten zur Verfügung gestellt werden, welche nacheinander durch verschiedene Einheiten fliessen, wobei jede Einheit eine bestimmte Operation auf den Daten durchführt. Pipeline-Architekturen haben den Vorteil, dass sie eine hohe Parallelisierung der Arbeit erlauben. Das heisst, dass alle Elemente der Pipeline gleichzeitig arbeiten können, solange am Anfang der Pipeline Daten zur Verfügung stehen. Der genaue Aufbau einer Rendering Pipeline kann von Hersteller zu Hersteller verschieden sein.

In Abbildung 6 wird als Beispiel die Rendering-Pipeline von DirectX gezeigt. Die Modelldaten („Vertex Data“ und „Primitive Data“) laufen dabei zuerst durch eine Tesselationseinheit. Diese Einheit wandelt wenn nötig die Inputdaten in Dreieckspolygone um. Dadurch wird garantiert, dass der Rest der Pipeline nur auf Dreieckspolygonen rechnen muss.

Danach werden die Daten entweder der „Fixed Function Pipeline“ oder der „Programmable Pipeline“ zur Verarbeitung übergeben. An dieser Stelle kann „Fixed Function Pipeline“ die Daten transformieren, beleuchten und/oder auf eine fest vorgegebene Art animieren. Im Gegensatz dazu hat der Programmierer in der „Programmable Pipeline“ durch Shader enorme Freiheiten und kann beliebige Algorithmen realisieren. Weil die Shader hier auf den Eckpunkten (Vertices) von Polygonen arbeiten, werden sie „Vertexshader“ genannt. In beiden Fällen, also der „Programmable Pipeline“ und der „Fixed Function Pipeline“, werden die übergebenen Dreieckspolygone vom 3-Dimensionalen Raum in Bildschirm-Koordinaten (Screen-Space) umgewandelt.

Darauf werden Daten, welche ausserhalb des Bildbereichs liegen, eliminiert (Clipping), Polygone, welche nicht dem Bildschirm zugewendet sind, werden entfernt (Back Face Removal), und die Polygone werden rasterisiert, das heisst in einzelne Bildpunkte (Pixel) aufgeteilt. Jeder Bildpunkt erhält dabei eine Interpolation der Daten der Eckpunkte, welche von deren Distanz zum Bildpunkt abhängig ist.

Diese Interpolierten Daten werden darauf wiederum entweder der „Fixed Function Pipeline“ oder der „Programmable Pipeline“ zur Verarbeitung übergeben. Dabei hat der Programmierer wiederum in der „Programmable Pipeline“ über einen Shader die volle Freiheit mit den interpolierten Eingabedaten zu realisieren was er möchte, während mit der „Fixed Function Pipeline“ nur die Texturierung und Einfärbung des Pixels realisiert werden kann. Auf dieser Ebene wird der Shader auch „Pixelshader“ genannt, da er auf einzelnen Pixeln operiert.

Zuletzt durchlaufen die fertig berechneten Pixel einerseits diverse Tests, welche verhindern könnten, dass der Pixel auf den Framebuffer gerendert wird (Apha Test, Depth Test, Stencil Test, Scissor Test), und andererseits wird eventuell die Farbe des Pixels nochmals durch gesetzte Nebel-Parameter (Fogging) verändert. Danach wird der Pixel mittels Alpha-Blending in den Framebuffer eingefügt. Der Framebuffer wird schlussendlich, sobald alle Modelle gerendert sind, auf dem Bildschirm angezeigt.

Eine exakte Erklärung und Definition der einzelnen Elemente der Pipeline kann in der Dokumentation von DirectX nachgeschlagen werden. An den zwei Stellen, bei welchen die Auswahl besteht, ob die „Fixed Function Pipeline“ oder die „Programmable Pipeline“ verwendet werden, liegt die Wahl beim Benutzer von DirectX, das heisst dem Programmierer. Für diese Studienarbeit relevant sind jeweils

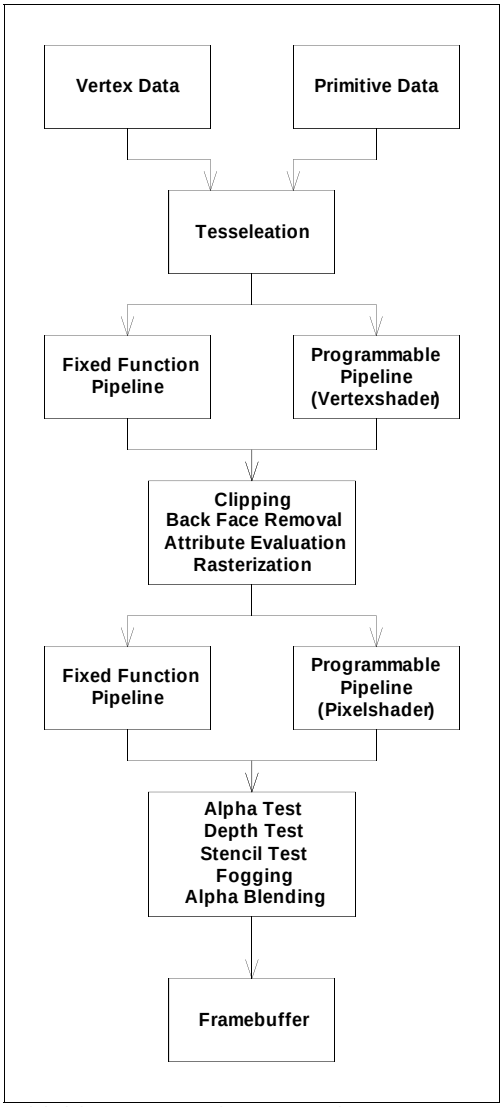


Abbildung 6: Rendering Pipeline

nur die Teile der „Programmable Pipeline“, auch wenn die Studienarbeit später theoretisch erweitert werden könnte, so dass sie auch noch die ältere „Fixed Function Pipeline“ unterstützt. Da die „Fixed Function Pipeline“ jedoch veraltet ist und sogar von Microsoft mit der Einführung von DirectX 10 aus dem DirectX SDK entfernt wird, ist nicht zu erwarten, dass noch Interesse an einer Umsetzung dafür besteht.

4 Vertex- und Pixelshader

Wie bereits im letzten Abschnitt gezeigt wurde, werden Shader in Vertex- und Pixelshader unterteilt. Diejenigen Shader welche auf den Eckpunkten (Englisch: Vertices) von Polygonen arbeiten, werden Vertexshader genannt und Shader welche auf den rasterisierten Pixeln arbeiten, werden Pixelshader genannt. Dadurch, dass die Shader auf unterschiedlichen Ebenen der Pipeline arbeiten, nehmen sie auch unterschiedliche Aufgaben wahr.

Die Vertexshader werden normalerweise für folgende Dinge eingesetzt:

- Animation der Modelle. Beispiele dafür sind Morphing ([WIKI06]) und Skinning ([WIKI07]). Manchmal werden auch auch die Positionen der Vertices direkt auf dem Vertexshader berechnet. Ein Beispiel dafür ist das sogenannte Displacement Mapping ([WIKI08]).

Die Pixelshader hingegen nehmen die folgenden Aufgaben wahr:

- Implementierung von verschiedenen Beleuchtungsmodellen. Beispiele dafür sind das Lambert Diffuse Lighting, Phong Specular Lighting und Anisotropic Lighting.
- Simulieren von unebenen Oberflächen. Beispiele dafür sind Normal Mapping und Parallax Occlusion Mapping.
- Weitere Algorithmen welche auf der Farbe oder der Tiefeninformation des Pixels basieren.

Dadurch dass die beiden Shaderarten für so verschiedene Aufgaben eingesetzt werden, war das Instruktionsset auch unterschiedlich. Beispielsweise konnten lange nur Pixelshader Texturen nutzen. Aus verschiedenen Gründen wäre es jedoch heute wünschenswert wenn beide Shaderarten das gleiche Instruktionsset besässen. Daher ist für die nahe Zukunft zu erwarten, dass Vertex- und Pixelshader das gleiche Instruktionsset erhalten werden.

5 Shader-Sprachen

5.1 Übersicht

Wie alle Prozessoren können Grafikprozessoren nur Maschineninstruktionen übersetzen. Das in Abbildung 7 gezeigte Statusdiagramm demonstriert, wie in DirectX aus dem Code einer Shaderhochsprache auf dem Grafikprozessor lauffähiger Maschinencode entsteht. Die durchlaufenen Stati sind die Folgenden:

g) Hochsprache: Der Shader implementiert in einer der verfügbaren Shaderhochsprachen. Dies könnten zum Beispiel HLSL, cg oder GLSL sein. Der Code wird mit dem Compiler des Herstellers der Hochsprache in Assemblercode umgewandelt.

h) Shader Assemblercode. Das Format des Assemblercodes und die unterstützten Assemblerinstruktionen werden durch DirectX spezifiziert. Sie sind Grafikkartenunabhängig. Um Grafikkarten mit unterschiedlichen Qualitäten zu unterstützen definiert DirectX verschiedene Assemblerversionen. Die momentan unterstützten Versionen sind: Vertexshader 1.1, 2.0, 2.0a, 3.0 und Pixelshader 1.1, 1.2, 1.3, 1.4, 2.0, 2.0a, 2.0b, 3.0. Der Assemblercode wird vom DirectX Assembler in DirectX Objectode übersetzt.

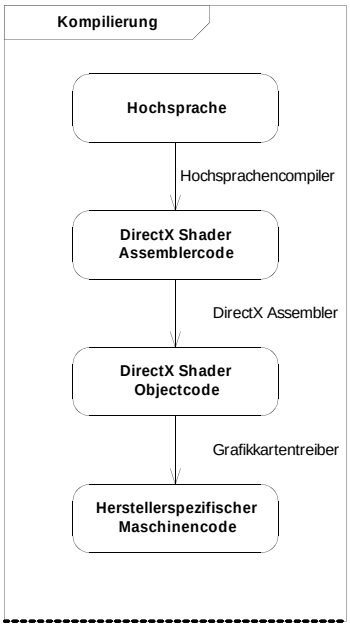


Abbildung 7: Shaderkompilierung

i) Shader Objectcode: Der Objectcode ist ein binärer Code. Dieser binäre Code wird von den Applikationen an die DirectX API übergeben. Das genaue Format des Codes ist unbekannt.

j) Herstellerspezifischer Maschinencode: Wenn Shader Objectcode an die DirectX API zur Ausführung übergeben wird, wird er von der API an den Grafikkartentreiber weitergeleitet. Der Grafikkartentreiber wiederum übersetzt den Objectcode in Maschinencode welchen die installierte Grafikkarte ausführen kann. Die meisten Treiber optimieren danach den Maschinencode nochmals, so dass er mit Geschwindigkeit auf der Grafikkarte laufen wird.

5.2 Assembler

Für den Applikationsentwickler ist der Shader Assemblercode die Basis der Shader. Als die Shader im Jahr 2000 mit DirectX 8 neu eingeführt wurden waren Shaderhochsprachen noch unbekannt und die Shader mussten in Assemblercode geschrieben werden. Codeabschnitt 1 und Codeabschnitt 2 zeigen den Assemblercode des Vertex- und Pixelshaderpaares welche aus dem im nächsten Abschnitt gezeigten HLSL Programm generiert worden sind.

```
vs_1_1
dcl_position v0
dcl_texcoord v1
mul r0, v0.y, c1
mad r0, c0, v0.x, r0
mad oPos, c2, v0.z, r0
mov oT0.xy, v1
```

Codeabschnitt 1: Vertexshader Assemblercode

```
ps_1_4
texld r0, t0
```

Codeabschnitt 2: Pixelshader Assemblercode

Die folgende Tabelle zeigt eine Übersicht über die von DirectX unterstützten Vertex- und Pixelshaderversionen:

DirectX Version	Vertexshader Versionen	Pixelshader Versionen
DirectX 8.0	1.0	1.0, 1.1
DirectX 8.1	1.1	1.2, 1.3, 1.4
DirectX 9.0	2.0	2.0
DirectX 9.0a	2.0a	2.0a
DirectX 9.0b		2.0b
DirectX 9.0c	3.0	3.0

5.3 HLSL

HLSL (Abkürzung für „High Level Shader Language“) ist die von Microsoft mit DirectX eingeführte Shaderhochsprache. Sie war nach cg von nVidia die zweite eingeführte Hochsprache und wurde im Dezember 2002 als Teil von DirectX 9.0 eingeführt.

Die Hochsprache besitzt viele Ähnlichkeiten zu C und ist zusätzlich einige spezielle Features mit denen sie auf die Bedürfnisse der Shaderprogrammierung angepasst worden ist. Beispiele dafür sind die Semantikangaben bei den Input-Argumenten und die Vektor- und Matrixdatentypen.

Das als Beispiel in Codeabschnitt 3 gezeigte HLSL-Programm texturiert ein Modell.

```
struct VS_OUTPUT
{
  float4 pos : POSITION;
  float2 texcoord : TEXCOORD0;
};

float4x4 matWorld;
sampler colorSampler;

VS_OUTPUT vs_main(
  float3 objPosition : POSITION0,
  float2 texcoord : TEXCOORD0
)
{
  VS_OUTPUT o;
  o.pos = mul(matWorld, objPosition);
  o.texcoord = texcoord;
  return o;
}

float4 ps_main(
  float2 texcoord : TEXCOORD0
) : COLOR0
{
  return tex2D(colorSampler, texcoord);
}
```

Codeabschnitt 3: HLSL Programm

Kapitel II: Domain Analyse

1 Einführung

Dieses Kapitel beschreibt die Objektorientierte Analyse von shaded. Als erstes wird das Konzept des Shadergraphen eingeführt. Im Weiteren wird das Domain Model von Shaded aufgeführt. Auf Systemsequenzdiagramme und die davon abgeleiteten Verträge (Contracts) wurde nach Absprache mit dem Betreuer verzichtet.

2 Shadergraph

Im Folgenden sollen einerseits die genaue Verwendung des Shadergraphen und andererseits die damit verbundenen Begriffe geklärt werden. Der Shadergraph visualisiert über eine spezielle Art von Datenflussdiagramm ein Programm welches Eingabedaten verarbeitet und diese verarbeiteten Daten danach wieder ausgibt. Die Eingabedaten werden über sogenannte Ausgabekanäle in den Graphen transportiert. Von diesen Ausgabekanälen fliessen die Daten über gerichtete Verbindungen zu Eingabekanälen von Verarbeitungselementen. Jedes Verarbeitungselement besitzt eine gewisse Anzahl dieser Eingabekanäle. Sobald ein Element auf allen Eingabekanälen Daten anliegen hat verarbeitet es die Daten auf eine spezifische Art und Weise und leitet die generierten Ausgabedaten auf alle seine Ausgabekanäle, wo sie über weitere Verbindungen zu weiteren Verarbeitungselementen weiterfliessen können. Am Schluss der Verarbeitung fliessen die Daten an die Eingabekanäle des Graphen und werden dadurch zu den Ausgabedaten des Graphen.

In Abbildung 8 wird eine konzeptuelle Darstellung eines solchen Shadergraphen gezeigt. Es zeigt ebenfalls, dass Verarbeitungselemente durch untergeordnete Graphen, sogenannnte Frames, dargestellt werden können. Dadurch wird es möglich, mehrrere Frames mit jeweils komplexen mathematischen Formeln zu einem Shadergraphen zusammenzufügen.

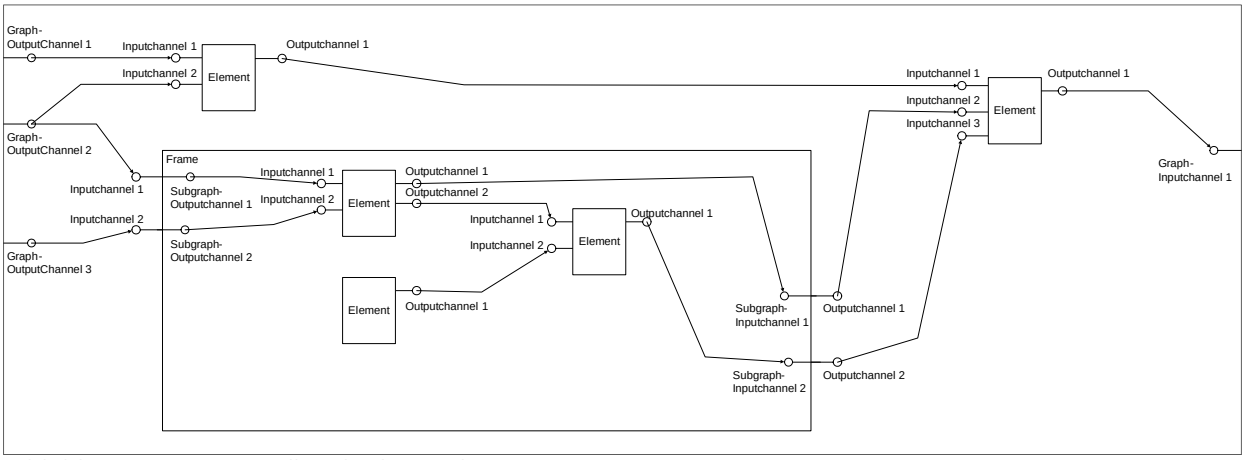


Abbildung 8: Konzeptueller Shadergraph

3 Domain Modell

3.1 Strukturdiagramm

Das grosse Strukturdiagramm von shaded wurde wie in [CLAR01] beschrieben in drei kleinere Strukturdiagramme und dadurch in drei Packages aufgeteilt. Die einzelnen Konzepte werden im nächsten Abschnitt gezeigt. Abbildung 9 bietet einen Überblick über die drei Packages von shaded.

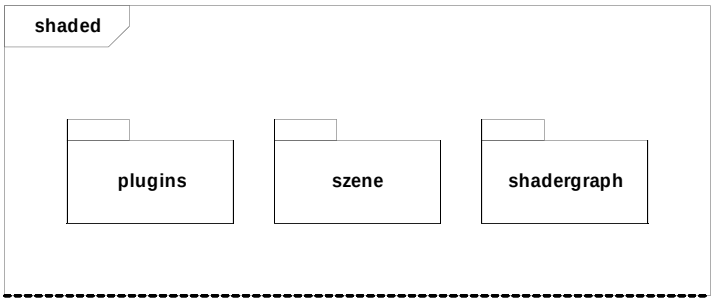


Abbildung 9: Strukturdiagramm Packages

Plugin-Package

Im ersten Package dem Plugin-Package wurden die drei Konzepte zu den verschiedenen verlangten Plugin-Interfaces untergebracht. Abbildung 10 zeigt dieses Package.

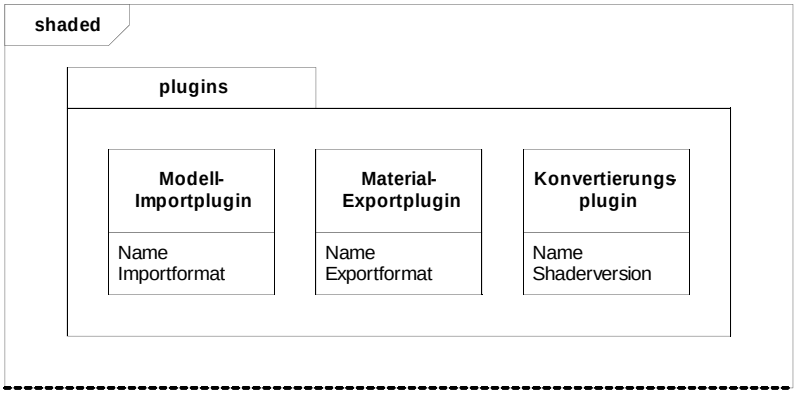


Abbildung 10: Strukturdiagramm Plugins

Szene-Package

Das zweite Package enthält alle Konzepte welche zu einer Szene gehören. Abbildung 11 zeigt dieses Package.

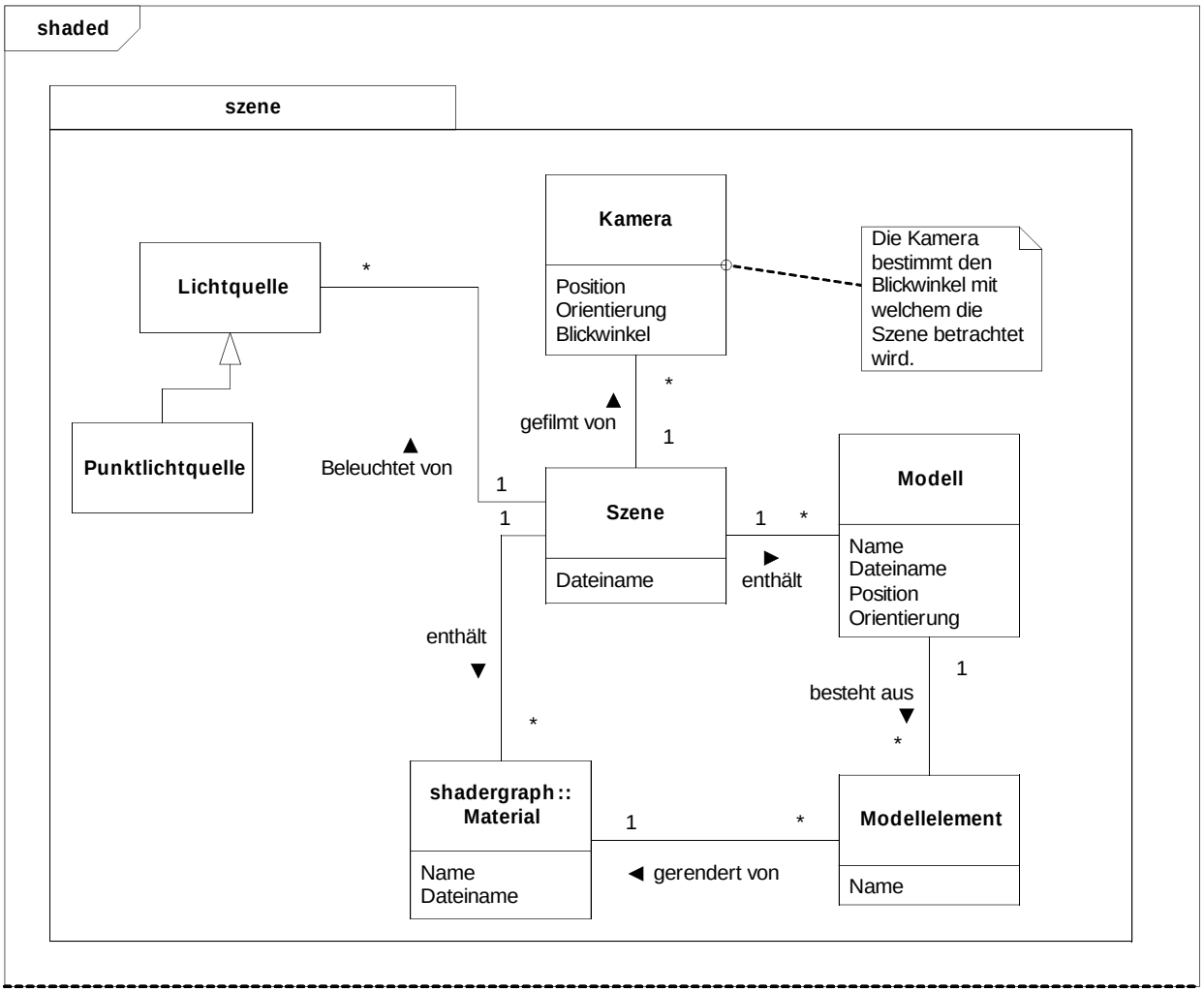


Abbildung 11: Strukturdiagramm Szene

Shadergraph-Package

Das dritte Package enthält alle Konzepte welche zur eigentlichen Aufgabenstellung von shaded, dem Shadergraph gehören. Abbildung 12 zeigt dieses Package.

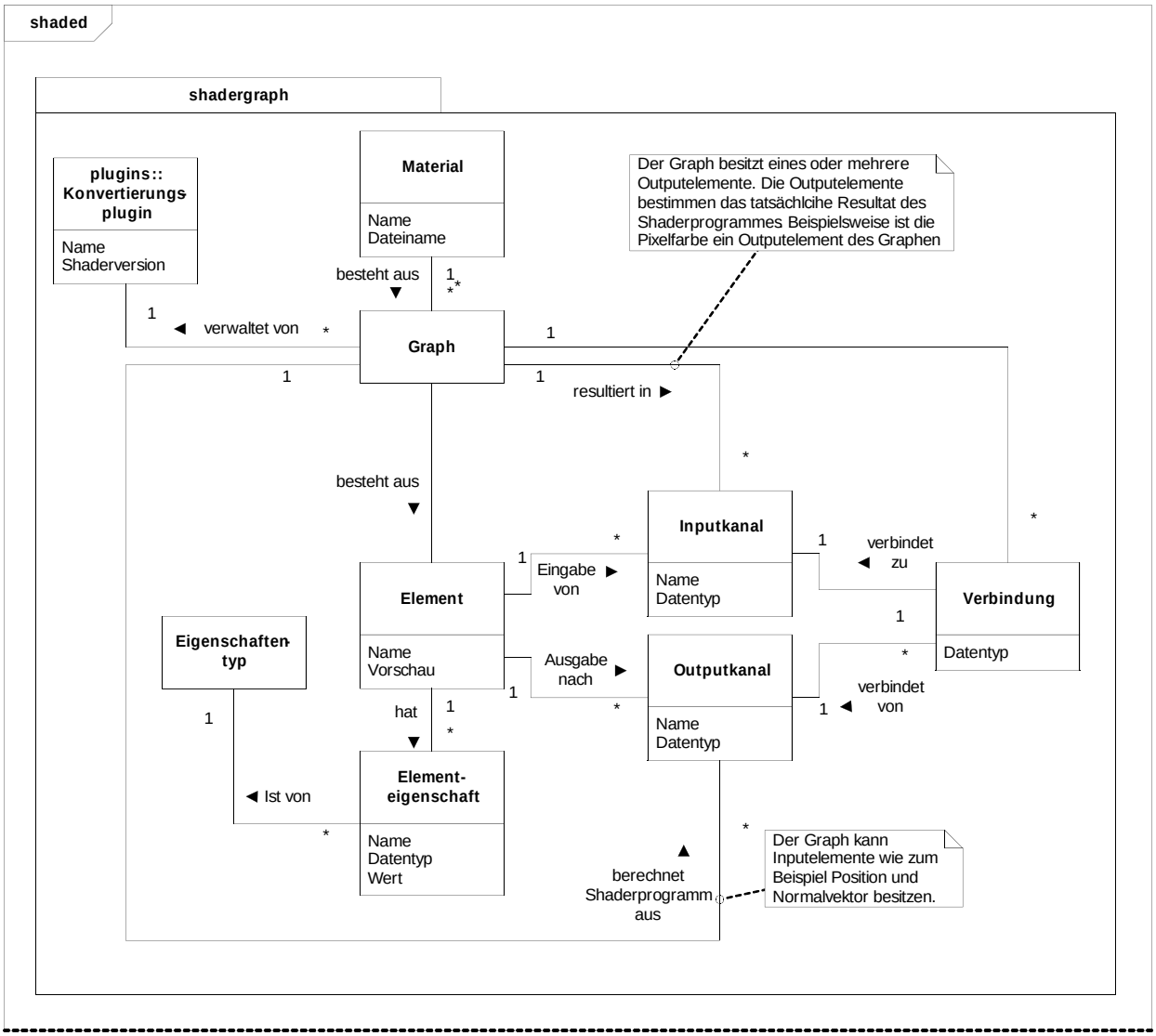


Abbildung 12: Strukturdiagramm Shadergraph

3.2 Konzeptbeschreibungen

Szene

Beschreibung Die Szene besitzt eine Ansammlung von Modellen, Materialien und Lichtquellen. Sie beschreibt den globalen Raum in welchem diese Elemente platziert und einander zugeordnet werden. Der Benutzer kann über eine Kamera eine Ansicht auf die Modelle erhalten.

Beziehungen Die Szene besitzt beliebig viele Modelle, Materialien, Lichtquellen und Kameras.

Attribute Das einzige Attribut der Szene ist der Dateiname. Dieses Attribut muss nicht gesetzt werden, wenn es aber gesetzt ist beschreibt es Pfad und Namen der Datei in welcher die Szene gespeichert ist.

Kamera

Beschreibung Eine Kamera beschreibt eine Ansicht bzw. den Blickwinkel auf die Szene. Das Konzept ähnelt einer Film- oder Fotokokamera.

Beziehungen Die Kamera hat eine Referenz auf die Szene welche sie filmt.

Attribute Die Kamera besitzt einen Namen und die Parameter, welche bestimmen wie die Szene aufgenommen wird. Dazu gehören Position, Orientierung und Blickwinkel (eine Art Zoomfaktor).

Lichtquelle

Beschreibung Lichtquellen beleuchten die Objekte in einer Szene. Sie sind notwendig weil sonst die ganze Szene nicht, bzw. nur von Umgebungslicht beleuchtet wäre und dadurch Modelle gar nicht oder nur an der Silhouette erkennbar wären.

Beziehungen Die Lichtquelle hat eine Referenz auf die Szene welche sie enthält.

Attribute Das primäre Attribut der Lichtquelle ist der Typ der Lichtquelle (Punktlichtquelle, Scheinwerfer, Paralleles Licht, Spezielle Lichtquelle). Je nach Typ besitzt die Lichtquelle weitere spezifische Attribute.

Modell

Beschreibung Ein Modell besteht aus einer Ansammlung von Modellelementen. Es organisiert die relative Positionierung und Orientierung zwischen diesen Unterelementen.

Beziehungen Das Modell hat eine Relation zur Szene in welcher es sich befindet und es besitzt die ihm untergeordneten Modellteile.

Attribute Das Modell besitzt einen Namen, welcher zur Identifizierung verwendet wird. Des weiteren besitzt es einen Dateinamen welcher immer gesetzt sein muss und auf die Datei zeigt aus welcher das Modell importiert worden ist. Ferner besitzt das Modell die Attribute Position und Orientierung, welche bestimmen wo sich das Modell in der Szene befindet und wie es darin orientiert ist.

Modellelement

Beschreibung Repräsentiert einen Teil eines Modells. Jedes Modellelement besitzt genau ein Material. Die Idee ist, dass ein Modell Unterelemente mit verschiedenen Materialien besitzen kann. Zum Beispiel kann das Modell eines Autos für die Karosserie ein anderes Material verwenden als für die Reifen. In diesem Beispiel wäre die Karosserie ein Modellelement und die Reifen ein anderes Modellelement.

Beziehungen Das Modellelement hat eine Referenz zum übergeordneten Modell und eine Referenz zum Material mit welchem es gerendert wird.

Attribute Das Modellelement hat als einziges Attribut einen Namen mit welchem es vom Benutzer identifiiziert werden kann.

Modell-Importplugin

Beschreibung Das Modell-Importplugin ist dafür verantwortlich Modelle zu laden.

Beziehungen Keine

Attribute Keine

Material-Exportplugin

Beschreibung Das Material-Exportplugin ist dafür verantwortlich Materialien zu exportieren.

Beziehungen Keine

Attribute Keine

Material

Beschreibung Das Material bestimmt das Aussehen eines Modelles. Es ist der Container welcher den Shadergraphen übergeordnet ist. Ein Material bestimmt das Aussehen eines Modellelementes.

Beziehungen Das Material hat eine Referenz auf die Szene in welcher es sich befindet. Zudem besitzt es eine beliebige Anzahl von Graphen wobei keine zwei Graphen mit demselben Konvertierungsplugin vorhanden sein müssen. Damit kann das Material Shadergraphen für verschiedene Shadersprachversionen unterstützen.

Attribute Das Material besitzt einen Namen mit welchem es vom Benutzer identifiziert werden kann. Zusätzlich besitzt es einen Dateinamen. Der Dateiname muss nicht gesetzt sein, wenn er aber gesetzt ist beschreibt er Pfad und Namen der Datei in welcher das Material gespeichert ist.

Graph

Beschreibung Das Konzept des Graphen hält die Elemente und Verbindungen zusammen. Diese Komponenten des Graphen stellen zusammen den Fluss der Daten im Shader und die Operationen auf denselben dar. Aus dem Graphen und seinen Elementen werden vom Konvertierungsplugin die Shader generiert. Er besitzt, je nach vom Konvertierungsplugin gegebenen Typ, eine gewisse Anzahl globaler Output- und Inputkanäle. Die Outputkanäle stellen dabei die für den Shader zur Verfügung stehenden Eingaben dar, während die Inputkanäle den Output der generierten Shader darstellen. (Die Namensgebung für die Kanäle stimmt nur in Bezug auf die Elemente des Graphen und nicht in Bezug auf den Graphen selbst.). Dadurch, dass der Graph einen Datenfluss darstellt ist gegeben, dass die Verbindungen immer eine Richtungsangabe enthalten müssen und keine Schlaufen entstehen dürfen. Die Richtung geht immer vom Outputkanal zum Inputkanal. Durch diese Vorgaben gilt, dass der Shadergraph ein DAG (Directed Acyclic Graph) ist.

Beziehungen Der Graph besitzt Referenzen auf sämtliche in ihm enthaltenen Elemente und Verbindungen. Zusätzlich besitzt er Referenzen auf seine Output- und Inputkanäle. Schliesslich hat er eine Referenz auf das Konvertierungsplugin mit welchem er erstellt worden ist.

Attribute Der Graph besitzt keine Attribute.

Konvertierungsplugin

Beschreibung Das Konvertierungsplugin bestimmt einerseits den Typ des Graphen, die dafür verwendbaren Elemente und Einschränkungen. Zusätzlich ist es dafür verantwortlich den Graphen in einen Shader der von ihm unterstützen Shadersprache zu konvertieren.

Beziehungen Das Konvertierungsplug hat eine 1:n-Relation zu den mit ihm erstellten Graphen.

Attribute Die Attribute des Konvertierungsplugins sind der Name mit welchem der Benutzer das Plugin identifizieren kann und die Shadersprache für welche das Plugin Shader generieren kann.

Element

Beschreibung Das Element ist ein Knoten im Graphen. Es hat für den Graphen keine weitere Bedeutung, bekommt sie jedoch bei der Interpretierung des Graphen durch das Konvertierungsplugin. Das Element ist jeweils eine bestimmte Komponente des zu generierenden Shaders.

Beziehungen Das Element hat einerseits eine Referenz auf den Graphen, zu welchem es gehört. Andererseits besitzt es Referenzen auf seine Eigenschaften, seine Inputkanäle und seine Outputkanäle.

Attribute Das erste Attribut des Elementes ist sein Name. Mit dem Namen können Benutzer und Konvertierungsplugin seinen Zweck identifizieren. Weiterhin besitzt es eine Vorschau-Ansicht, welche zeigt, wie das Resultat des resultierenden Shaders bei diesem Element aussehen wird.

Elementeigenschaft

Beschreibung Elemente des Graphen können Eigenschaften besitzen. Eigenschaften können zum Beispiel beeinflussen wie ein Element vom Konvertierungsplugin in Shadercode umgesetzt wird.

Beziehungen Jede Eigenschaft hat eine Referenz auf das Element zu welchem sie gehört und eine Referenz auf das Eigenschaftentyp-Konzept welches den Typ dieser Eigenschaft beschreibt.

Attribute Jede Eigenschaft besitzt zwei Attribute. Der Name dient dazu, dass die Eigenschaft vom Benutzer identifiziert werden kann und Wert-Attribut speichert den von der Benutzeroberfläche gemäss dem Eigenschaftentyp gesetzten Wert.

Eigenschaftentyp

Beschreibung Die Elementeigenschaften können unterschiedliche Typen haben. Dieses Konzept soll betonen, dass es wichtig ist zwischen den verschiedenen Typen von Eigenschaften zu unterscheiden.

Beziehungen Keine

Attribute Keine

Inputkanal

Beschreibung Der Inputkanal stellt einen Verknüpfungspunkt im Graphen dar. Der Graph stellt schlussendlich den Programmfluss dar und der Inputkanal bestimmt, welche Daten an das Element, zu welchem der Inputkanal gehört, übergeben werden. Jeder Inputkanal kann nur eine Verbindung gleichzeitig erhalten.

Beziehungen Der Inputkanal besitzt entweder eine Referenz auf das Element, zu welchem er gehört oder, wenn er zum Graphen gehört, eine Referenz auf den Graphen.

Attribute Jeder Inputkanal besitzt einen Namen über welchen der Benutzer den Zweck des Kanals erkennen kann. Zusätzlich besitzt der Inputkanal eine Angabe des Typs der Daten, welche von ihm angenommen werden können. Mit dieser Angabe kann sichergestellt werden, dass Outputkanäle nur mit vom Typ her kompatiblen Inputkanälen verbunden werden.

Outputkanal

Beschreibung Der Outputkanal stellt einen Verknüpfungspunkt im Graphen dar. Der Graph stellt schlussendlich den Programmfluss dar und der Outputkanal bestimmt welche Daten von einem Element ausgegeben werden. Diese Daten können dann über Verbindungen an die Inputkanäle anderer Elemente weitergegeben werden. Jeder Outputkanal kann seine Daten über beliebig viele Verbindungen weitergeben.

Beziehungen Der Outputkanal besitzt entweder eine Referenz auf das Element, zu welchem er gehört oder, wenn er zum Graphen gehört, eine Referenz auf den Graphen.

Attribute Jeder Outputkanal besitzt einen Namen, über welchen der Benutzer den Zweck des Kanals erkennen kann. Zusätzlich besitzt der Outputkanal eine Angabe des Typs der Daten welche von ihm ausgegeben werden. Mit dieser Angabe kann sichergestellt werden, dass Outputkanäle nur mit vom Typ her kompatiblen Inputkanälen verbunden werden.

Verbindung

Beschreibung Die Verbindung stellt den Datenfluss zwischen einem Outputkanal und einem Inputkanal dar. Der Datenfluss funktioniert dabei so, dass die Daten vom Outputkanal zum Inputkanal fliessen.

Beziehungen Die Verbindung hat eine Referenz auf den Graphen, zu welchem sie gehört, eine Referenz auf den Outputkanal von welchem die Daten stammen und eine Referenz auf den Inputkanal, zu welchem die Daten fliessen sollen.

Attribute Das einzige Attribut einer Verbindung ist die Angabe welcher Typ von Daten über die Verbindung fliessen wird.

Kapitel III: Übersicht Design

1 Einführung

Dieses Kapitel soll verschiedene Aufgaben erfüllen. Die wichtigste davon ist, einen leichten Einstieg in das Design von Shaded zu ermöglichen. Es zeigt auf wie die Software in einzelne Komponenten und Schichten aufgeteilt wurde. Zusätzlich werden die wichtigsten Konzepte eingeführt, auf welchen die folgenden Kapitel aufbauen.

Des Weiteren soll dieses Kapitel aber auch einen kurzen Überblick darüber geben, wie das Design zustande gekommen ist, und wie es sich bis zum endgültigen, dokumentierten Zustand entwickelt hat.

2 Logische Architektur

Die Komponenten von Shaded wurden in drei verschiedene Layer aufgeteilt. Dadurch wurde eine klassische 3-Layer-Architektur mit einem Presentation-, einem Domain- und einem Foundation-Layer angestrebt. Abbildung 13 zeigt diese Aufteilung. Im Detail umfassen die Layer von shaded die folgenden Funktionen:

- Foundation-Layer: Das Foundation-Layer wird zum grössten Teil von der .NET-Code-Bibliothek abgedeckt. Für Shaded wurden zusätzliche Klassen für Logging in Dateien und für Mathematik hinzugefügt. Die Klassen des Foundation-Layer könnten bei beliebigen Projekten wiederverwendet werden.
- Domain-Layer: Das Domain-Layer implementiert die Domain-Spezifischen Elemente von Shaded. Dazu gehören der Shadergraph, die Scene- und Rendering-Klassen. Die Klassen des Domain-Layers implementieren die „Business“-Logik und könnten in einer ähnlichen Applikation mit einer anderen Benutzeroberfläche wiederverwendet werden.
- Presentation-Layer: Das Presentation-Layer realisiert die Benutzeroberfläche von Shaded.

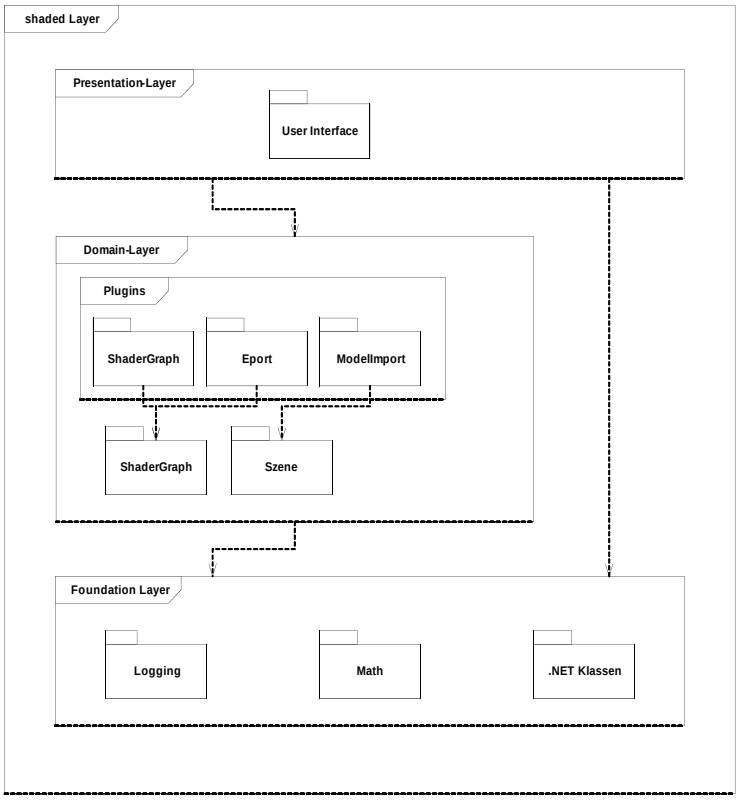


Abbildung 13: Layer

3 Die wichtigsten Elemente

Der vorliegende Abschnitt soll eine Übersicht über die wichtigsten Elemente von shaded geben. Damit wird ein leichter Einstieg in das Design der Software gegeben und es kann von diesen zentralen Punkten ausgehend in den nachfolgenden Kapiteln intensiver betrachtet werden.

3.1 Die Klasse Program

Die Klasse Program ist der Einstiegspunkt und das Bindeglied der Software. Sie konstruiert dazu die wichtigsten Elemente der Benutzeroberfläche und enthält die Referenz auf die von der Benutzeroberfläche verwendete Szene. In Abbildung 14 wird das Klassendiagramm gezeigt.

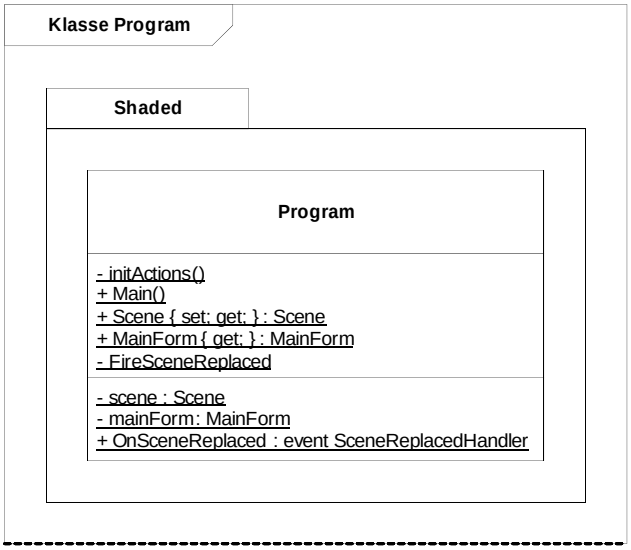


Abbildung 14: Klasse Program

In der Main-Methode wird das MainForm der Benutzeroberfläche initialisiert und in einem statischen Attribut gespeichert. Über das ebenfalls statische MainForm-Property kann später von überall her darauf zugegriffen werden. Diese Funktionalität wird für die in Kapitel (siehe Kapitel V) gezeigte Aktionsbehandlung benötigt.

Des weiteren wird während der Ausführung der Main-Methode eine Instanz einer Szene erstellt. Mit dieser leeren Szene wird die Benutzeroberfläche initialisiert. Alle Elemente der Benutzeroberfläche greifen immer über das Scene-Property auf diese Referenz zurück. Falls eine Aktion eine neue Szene auf der Benutzeroberfläche anzeigen möchte, so kann sie das Scene-Property ändern, worauf über das Auslösen des OnSceneReplaced-Events die Benutzeroberfläche über die Änderung benachrichtigt wird.

3.2 Die Klasse PluginManager

Wie in der Analyse gezeigt (siehe Kapitel II) verwendet shaded drei verschiedene Arten von Plugins. Die in Abbildung 15 gezeigte Klasse PluginManager bietet einen zentralen Punkt, um auf alle geladenen Plugins zuzugreifen. Beim ersten Zugriff auf ein Element der Klasse wird im statischen Konstruktor nach Plugin-Implementierungen gesucht. Dafür werden alle .NET-Assemblies welche sich im gleichen Verzeichnis befinden wie die .exe-Datei von shaded nach Klassen, welche eines der Plugin-Interfaces implementieren, durchsucht. Danach kann auf die gefundenen Plugins über eine der drei Properties ShadergraphPlugins, ExportPlugins und ModelImportPlugins zugegriffen werden.

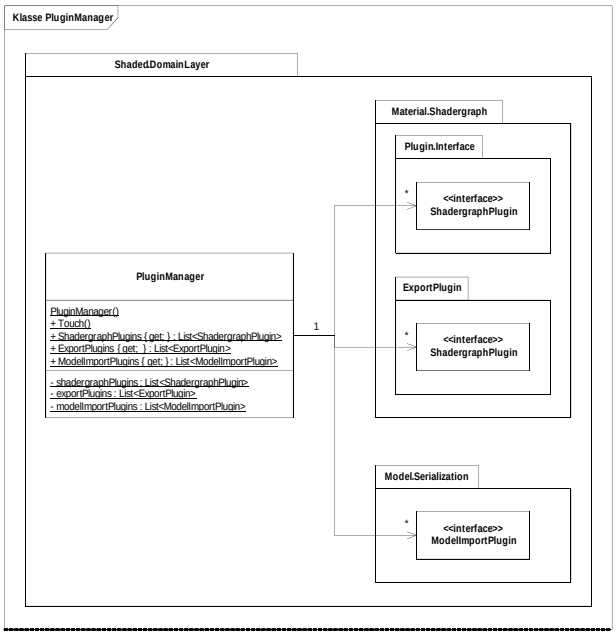


Abbildung 15: Klasse PluginManager

3.3 Plugin Interfaces

Im letzten Abschnitt wurde der Plugin Manager vorgestellt. In diesem Abschnitt sollen die Interfaces zu den drei Arten von Plugins vorgestellt werden.

Das Shadergraph-Plugin-Interface

Das Shadergraph-Plugin-Interface ist das komplexeste und auch das wichtigste Plugin-Interface von shaded. Ein grosser Teil der Logik der Problem Domain wird von der eigentlichen Problem Domain in die Implementierungen des Plugin-Interfaces verschoben. Die Abbildung 16 zeigt eine stark vereinfachte Version des Klassendiagrammes des Shadergraph-Plugin-Interfaces. Die detaillierte Version ist aus Platzgründen im Anhang zu finden.

Die Klassenstruktur ist abgeleitet vom entsprechenden Domain Model (siehe Kapitel II Abschnitt 3.1) und ihm daher sehr ähnlich. Die grossen Unterschiede im Vergleich zum Domain Model bestehen in den Klassen, welche zur Kommunikation mit der Szene benötigt werden und in den Frame-Klassen welche im Domain Model noch nicht aufgeführt sind. Die in einem späteren Abschnitt vorgestellte Szene liefert über das interface `SceneInterface` die vom Shadergraphen benötigten Eingabewerte. Zum Zeitpunkt der Studienarbeit bestehen die Eingabewerte nur aus allen vom Benutzer definierten und durch den Shadergraphen verwendbaren Lichtquellen. Das Resultat des Shadergraphen wird in Implementierung des Interfaces `CompiledOutput` gespeichert. Dieses Interface enthält zusätzlich zum generierten Vertex- und Pixelshaderpaar Angaben über die vom Shadergraphen verwendeten Textureinheiten und konstanten Eingabewerte, welche von der Szene bereitgestellt werden müssen.

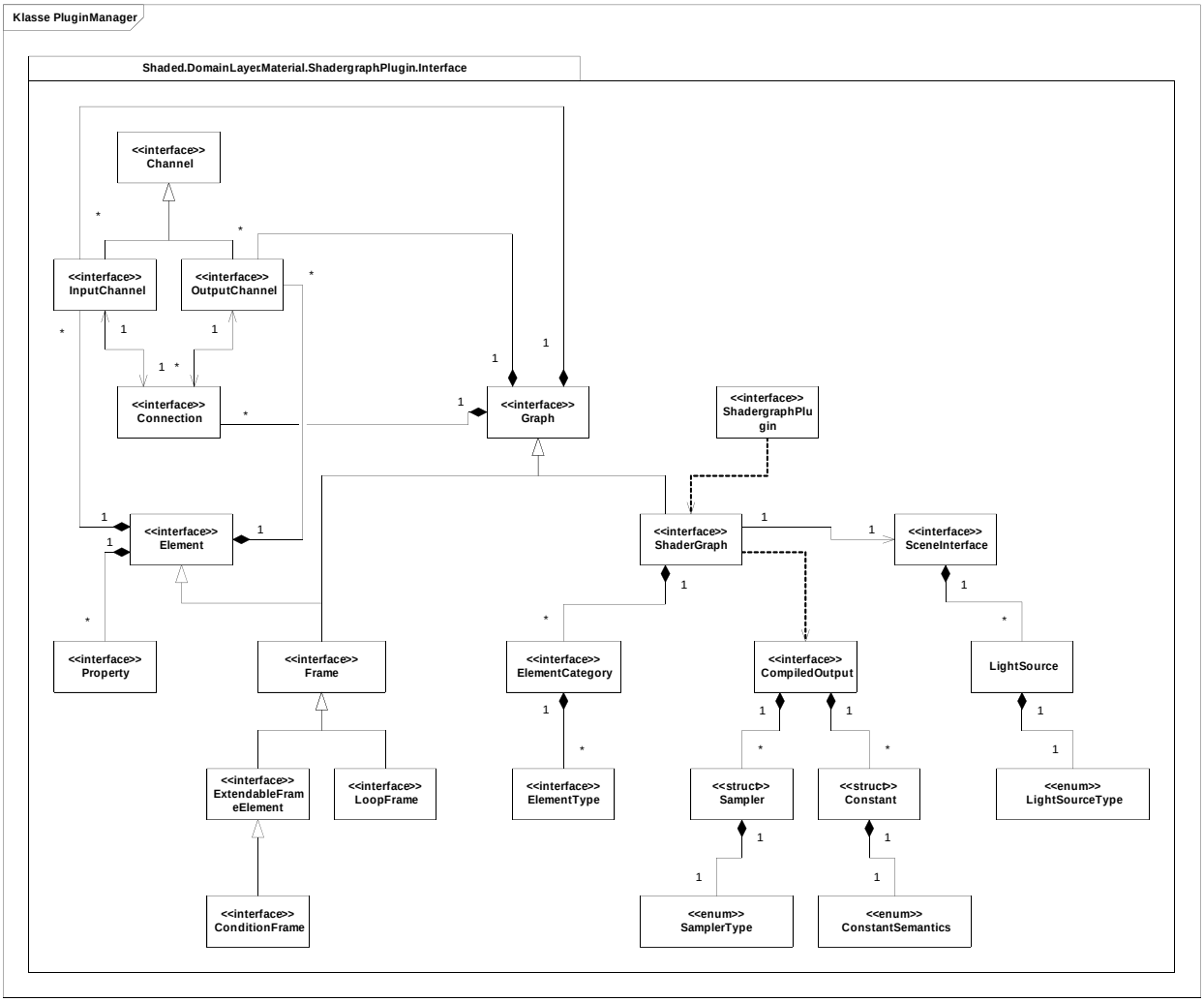


Abbildung 16: Shadergraph Plugin

Das Modelimport-Plugin-Interface

Implementierungen dieses Plugins werden verwendet, um 3-Dimensionale Modelle aus Dateien zu laden. Die genaue Definition eines Modelles folgt im Kapitel über die Szene (siehe Kapitel VI). Da Modelldateiformate normalerweise sehr komplex sind, könnte das Interface des Import-Plugins ebenfalls beliebig komplex gestaltet sein. Für die von shaded benötigten Zwecke kann das Design jedoch einfach gehalten werden. Die Abbildung 17 zeigt das Klassendiagramm.

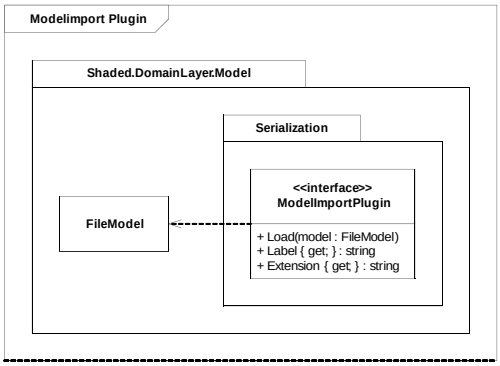


Abbildung 17: Modelimport Plugin

Das Plugin erhält über die Methode Load das Kommando, ein Modell zu laden. Als Argument wird ihr ein bereits initialisiertes FileModel übergeben. Von diesem Objekt kann das Plugin den Pfad der zu ladenden Datei erhalten. Danach soll das Plugin das Modell entsprechend dem später vorgestellten Model-Interface laden.

Das Export-Plugin-Interface

Dieser Plugin-Typ wird dazu verwendet, die erstellten Shader so zu speichern, so dass der Benutzer von shaded sie in seinem Produkt verwenden kann. Da das Resultat eines Shadergraphen jedoch stark vom Shadergraph-Plugin abhängig ist, sind Export-Plugins jeweils von einem oder mehreren spezifischen Shadergraph-Plugins abhängig.

Die Funktionsweise ist einfach. Das Export-Plugin erhält eine Referenz auf den CompiledOutput des Shadergraphen und eine Referenz auf einen Stream, in welchen es dann das exportierte Format schreiben kann. Die Abbildung 18 zeigt das dafür benötigte Klassendiagramm.

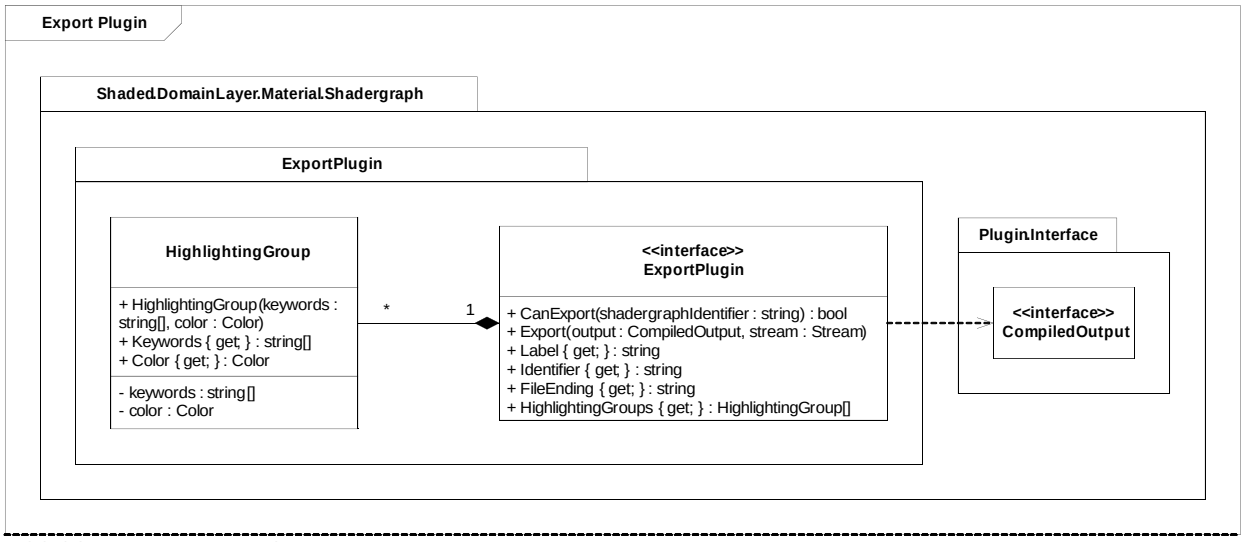


Abbildung 18: Export Plugin

3.4 Shadergraph DomainLayer

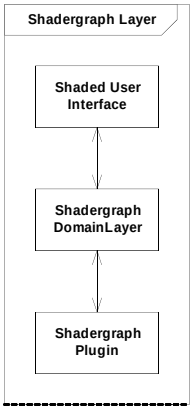


Abbildung 19: Layer

Wie im oben stehenden Abschnitt beschrieben werden in einem Shadergraph Plugin keine Informationen gespeichert welche zur Darstellung des Shadergraphen benötigt würden. Diese Informationen werden in einem speziellen Softwarelayer gespeichert, welches das Plugin kapselt. In der Abbildung 19 wird diese Kapselung visualisiert.

Dadurch, dass die kapselnde Schicht nur zwischen den fundamentalen Differenzen zwischen normalen Elementen und den verschiedenen Arten von Frames unterscheiden muss vereinfacht sich Design im Vergleich zum Shadergraph Plugin. Diese Vereinfachung ermöglicht es, Elemente und die verschiedenen Kategorien von Frames zu generalisieren. Zusätzlich wurde eine Möglichkeit gesucht, sämtliche Elemente der Benutzeroberfläche von einem einheitlichen Interface abzuleiten. Dieses interface wurde in Form des Interfaces `InteractiveComponent` gefunden. Dieses gemeinsame interface erlaubt es, Objekte in Bezug auf ihre Erscheinen auf der Benutzeroberfläche einheitlich zu behandeln und zu manipulieren. Die Abbildung 20 zeigt eine stark vereinfachte Version des Klassendiagrammes. Die detaillierte Form wurde aus Platzgründen in den Anhang verschoben.

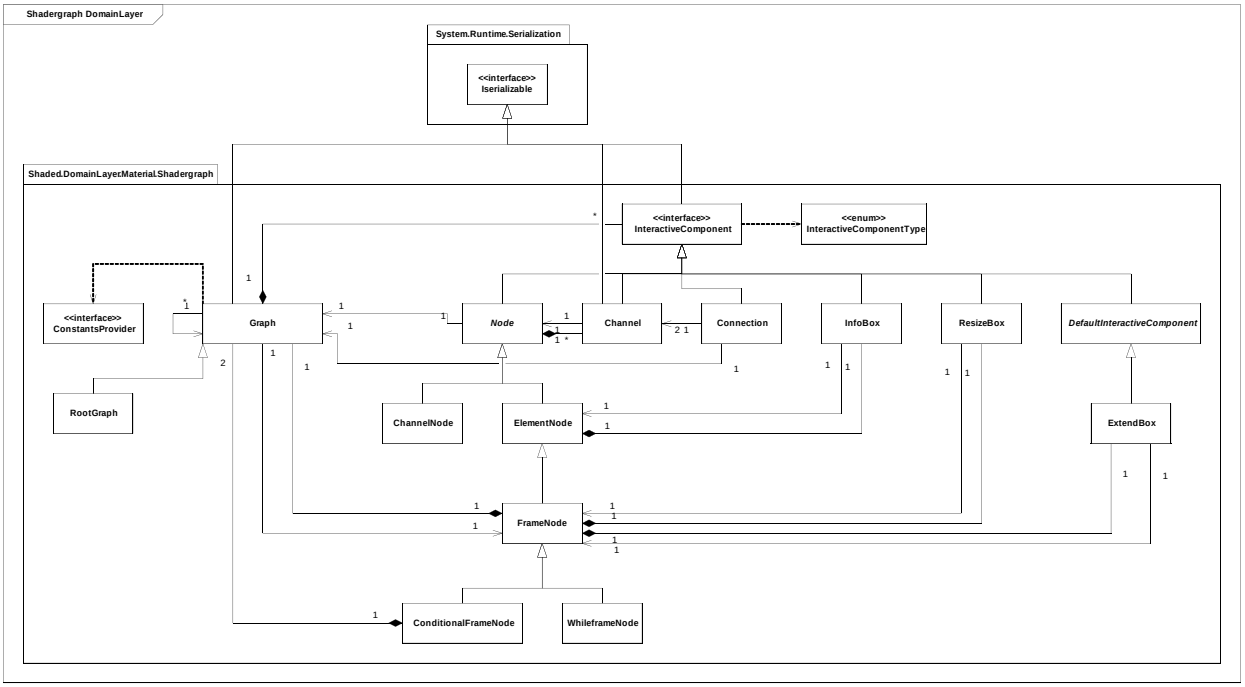


Abbildung 20: Shadergraph DomainLayer

Eine wichtige Funktion des Interfaces `InteractiveComponent` und der Klasse `Graph` ist die interne Ordnung der Objekte. Aus dem detaillierten Klassendiagramm ist ersichtlich, dass die Klasse RootGraph eine Referenz auf den Shadergraphen besitzt. Des weiteren kann aus dem Klassendiagramm gelesen werden, dass jeder Graph eine beliebige Anzahl von Komponenten (InteractiveComponent) besitzen kann. Mit diesen Komponenten kann der Benutzer interagieren. Einzelne dieser Komponenten, beispielsweise die Klassen `FrameNode`, `ConditionalFrameNode` und `WhileframeNode`, können wiederum weitere Graphen, sogenannte Subgraphen enthalten. Dadurch können Graphen beliebig ineinander verschachtelt und so eine hierarchische Struktur gebildet werden. Die Klasse RootGraph bildet dabei die Wurzel und den kontrollierenden Ursprung der ganzen Struktur. Die Abbildung 21 visualisiert ein einfaches Beispiel einer solchen hierarchischen Verschachtelung von Komponenten und Graphen anhand einer Baumstruktur.

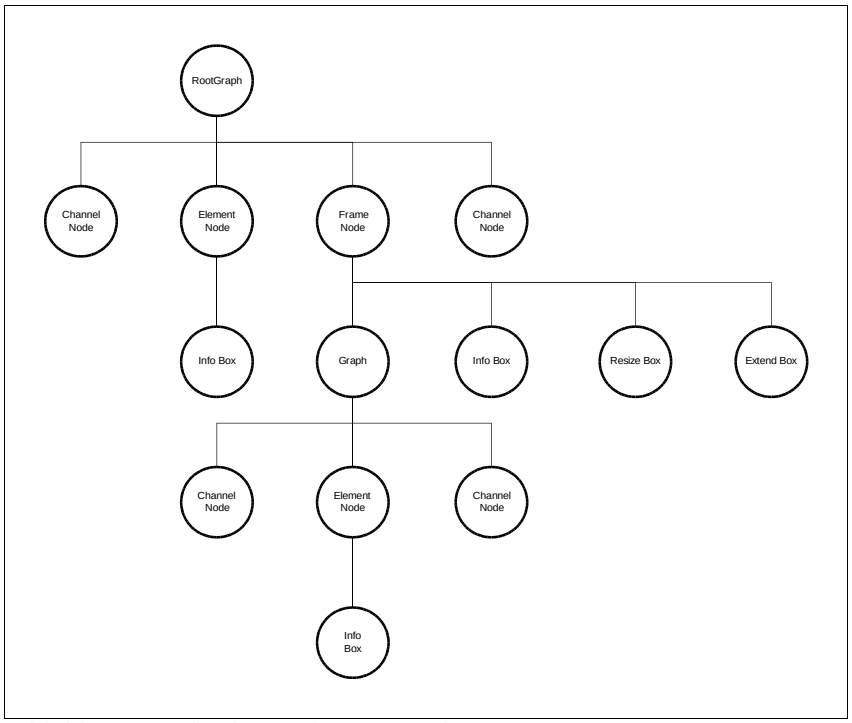


Abbildung 21: Shadergraph Baumstruktur

Eine zweite wichtige Eigenschaft des Interfaces `InteractiveComponent` ist, dass es dadurch möglich ist die Interaktion des Benutzers mit den Komponenten eines Graphen zu vereinheitlichen. Die Klasse Graph muss sich dadurch nie darum kümmern, ob eine Komponente ein Element des Shadergraphen, eine Verbindung zwischen zwei Elementen oder ein Hilfsobjekt (Beispielsweise eine Resize-Box) ist. Daher können in der Klasse Graph drei einfache Algorithmen zur Manipulation der Komponenten implementiert werden. Diese drei Algorithmen sind:

- Hervorheben von Komponenten: Dadurch wird jeweils diejenige Komponente mit welcher der Benutzer interagieren kann grafisch hervorgehoben.
- Selektieren von Objekten: Mittels diesem Algorithmus wird bestimmt, welche Komponente selektiert wird, wenn der Benutzer auf den Shadergraphen klickt.
- Verschieben/Dragging von Objekten: Durch diesen Algorithmus können Objekte verschoben werden. Beispielsweise kann die Resize-Box eines Frame-Elementes verschoben werden, wodurch sich die Grösse des Frames entsprechend der Verschiebung ändert.

Da alle drei Algorithmen einander sehr ähnlich sind wird an dieser Stelle als Beispiel im Codeabschnitt nur der Algorithmus zum Hervorheben von Objekten gezeigt. Er funktioniert ungefähr folgendermassen:

1: Für die Prioritäten 1 und 2 gehe durch alle Komponenten des Graphen

1.1: Teste Für jede Komponente ob die Priorität passt, wenn nicht übergehe die Komponente.

1.2: Berechne die Distanz zwischen Komponente und Mauszeiger. Teste danach ob die Distanz kleiner als die Highlighting-Distanz der Komponente ist. Falls ja; Teste ob die Distanz kleiner ist als die übergebene minimale Highlightingdistanz, so dass diejenige Komponente hervorgehoben wird, welche sich am nächsten beim Mauszeiger befindet.

1.3: Wenn dieser Test ebenfalls zugunsten der aktiven Komponente ausgefallen ist, unterscheide ob die Komponente ein Container ist und Kind-Komponenten besitzt oder ob es eine hervorhebbare Komponente ist.

1. Falls das Element ein Container ist, dann führe die Schritte 1.1 bis 1.3 für diese Komponente durch.
2. Falls die Komponente hervorhebbar ist dann wähle sie als aktuell hervorgehobene Komponente und überschreibe die aktuelle minimale Distanz mit derjenigen zwischen der Komponente und dem Mauszeiger.

```
DoHighlightTests(
  priority,
  component,
  ref highlightedComponent,
  ref distanceSqr,
  mousePoint
)
{
  if (component.HighlightPriority != priority)
    return

  currentDistanceSqr = component.GetDistanceSqr(mousePoint)
  if (component.HighlightTest() &&
      currentDistanceSqr < component.HighlightDistance ^ 2 &&
      currentDistanceSqr < distanceSqr)
  {
    if (component.Children.Count > 0)
    {
      foreach (child in component.Children)
      {
        DoHighlightTests(
          priority, child,
          ref highlightedComponent, ref distanceSqr,
          mousePoint
        )
      }
    }
    else
    {
      distanceSqr = currentDistanceSqr
      highlightedComponent = component
    }
  }
}

DoHighlightTests(Point mousePoint)
{
  for (priority = 1; priority < 3; ++priority)
  {
    InteractiveComponent highlightedComponent = null
    float distanceSr = MaxFloat
    foreach (component in GraphComponents)
    {
      DoHighlightTests(
        priority, component,
        ref highlightedComponent, ref distanceSqr, mousePoint
      )
    }
    if (highlightedComponent != null)
    {
      HighlightedComponent = highlightedComponent;
      break;
    }
  }
}
```

Codeabschnitt 1: Algorithmus zum Hervorheben von Komponenten (vereinfachter C#-Code)

3.5 Shadergraph-Material

In einem späteren Kapitel (siehe Kapitel VI) wird die Szene vorgestellt. Sie ist der Container, welcher einerseits sämtliche Daten des DomainLayers von shaded zusammenhält und andererseits das Rendering in der Vorschau-Ansicht übernimmt. Für das Aussehen der Oberfläche von Modellen sind dabei die Materialien der Szene wichtig. Sie besitzt vordefinierte Materialien um zum Beispiel die Lichtquellen oder lokale Koordinatensysteme der Szene zu visualisieren. Daneben existiert jedoch das Shadergraph-Material. Wie der Name bereits andeutet, besitzt das Shadergraph-Material einen Rootgraphen und damit indirekt einen Shadergraphen. Die Benutzeroberfläche greift auf die Shadergraphen immer über die Klasse `ShadergraphMaterial` zu. Daher besitzt diese Klasse wie auch die Klassen `ShaderGraph` im Plugin-Interface und `RootGraph` im DomainLayer eine zentrale Bedeutung.

Kapitel IV: shaded Kern

1 Einführung

Dadurch, dass der grösste Teil der Kernklassen durch das .NET-Framework bereitgestellt wird, ist der Kern von shaded sehr klein. Er besteht aus Klassen zur Verbesserung der Log-Ausgaben und einigen Klassen für mathematische Zwecke.

2 Logging

Wenn Informationen über das übliche `Console.WriteLine()` geschrieben werden, gelangen sie nur auf die Standardausgabe. Normalerweise, vor allem nach der Auslieferung des Produktes, sollen diese Ausgaben statt in eine Konsole in eine Datei geschrieben werden. Dazu müsste die Applikation jedoch die Standardausgabe in eine Datei umleiten. Insbesondere während dem Debugging der Applikation ist es jedoch unerlässlich, dass die Programmausgaben auch auf die Konsole ausgegeben werden. Daher ist es nicht wünschenswert, dieses Problem über eine Umleitung der Standardausgabe zu erreichen. Statt dessen bietet das .NET-Framework im `System.Diagnostics`-Namensraum eine Lösung für dieses Problem: Die Klasse `Trace`.

Diese Klasse kann wie die Klasse `Console` verwendet werden. Sie gibt die Ausgaben jedoch nicht an eine vordefinierte Stelle aus, sondern es ist möglich, sogenannte Listener zu registrieren. Die Klasse `Trace` leitet danach alle Ausgaben, welche sie erhält, an alle registrierten Listener weiter. Shaded verwendet zur Lösung des oben genannten Problemes zwei solche Listener. Einer schreibt die Programmausgaben für das Debugging der Applikation auf die Konsole und ein weiterer schreibt die Ausgaben in eine Logdatei. Die Abbildung 22 zeigt das Klassendiagramm:

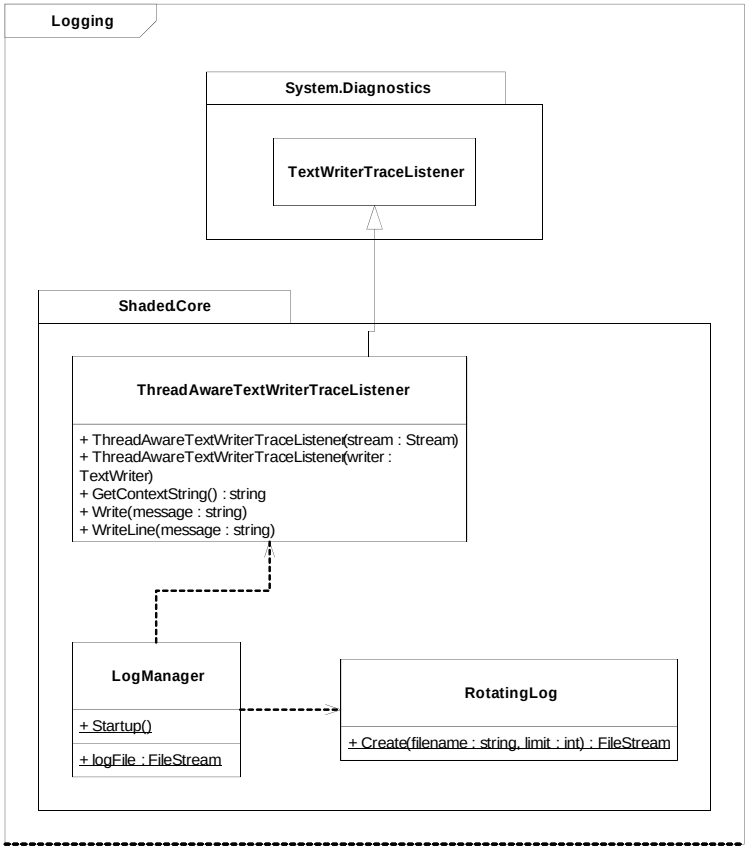


Abbildung 22: Logging

3 Mathematik-Klassen

Als grafische Applikation mit 3-Dimensionalen Elementen benötigt shaded diverse Klassen zur Umsetzung von mathematischen Konzepten. Diese Klassen werden jedoch nicht durch das .NET-Framework bereitgestellt. Ein Teil der für shaded benötigten Klassen sind im DirectX-SDK implementiert und könnten von dort verwendet werden. Da jedoch einerseits das DirectX-SDK nicht alle Ansprüche deckt und daher sowieso Ergänzungen nötig wären und andererseits bereits Referenzimplementierungen aus anderen Projekten vorhanden waren, wurde beschlossen, die nötige Funktionalität im shaded Kern selbst zu implementieren. Die folgende Aufzählung beschreibt die im Namensraum `Shaded.Core.Math` implementierten Klassen:

- *AlignedBox3:* Diese Klasse implementiert eine sogenannte Axis-Aligned-Box. Eine solche Box ist immer am lokalen Koordinatensystem ausgerichtet. Zur Beschreibung werden daher nur die Verschiebung der Box bezüglich des Ursprungs und die Dimensionen der Box, das heisst Höhe, Breite und Tiefe benötigt. Die Klasse implementiert diese Eigenschaften. Zusätzlich enthält sie einen Konstruktor, welcher aus einem Set von Punkten die kleinste alle Punkte umfassende Box berechnet.
- *Color:* Die Klasse implementiert eine normale ARGB-Farbe. Anders als jedoch bei den meisten Implementationen wird für jeden Farbkanal eine Fliesskommazahl verwendet. Dadurch erhält die Klasse eine erhöhte Genauigkeit bei Berechnungen. Die Klasse implementiert die Farbkanäle und eine Umwandlung in einen integralen 32-Bit-Farbwert.
- *Distance:* Diese Klasse bietet eine Ansammlung von statischen Methoden zur Distanzberechnung.
- *Matrix4x4f:* Diese Klasse implementiert eine Matrix mit vier Spalten und vier Zeilen. Angeboten werden einerseits Methoden um Matrizen für verschiedene Zwecke zu erstellen und andererseits

Methoden für mathematische Operationen mit Matrizen.

- *Quaternion:* Ein Quaternion wird in der Computergrafik oft verwendet, um Rotationen zu speichern und mit ihnen zu rechnen. Diese Klasse implementiert die dafür nötigen Methoden.
- *Sphere3:* Diese Klasse implementiert die Eigenschaften einer Kugel im Raum.
- Vector2f / Vector2i: Diese Klassen implementieren die Eigenschaften und Operationen auf einem 2-Dimensionalen Vektor. Sie unterscheiden sich im Datentyp der x- und y-Komponenten des Vektors. Während der Vector2f 32-Bit Fliesskommazahlen verwendet werden vom Vector2i integrale 32-Bit Zahlen verwendet.
- Vector3f: Diese Klasse implementiert die Eigenschaften und Operationen auf einem 3-Dimensionalen Vektor. Für die x-, y- und z-Komponente werden jeweils 32-Bit Fliesskommazahlen verwendet.

Kapitel V: User Interface

1 Einführung

Dieses Dokument zeigt im ersten Teil wie die Benutzeroberfläche anhand einer Analyse verwandter Produkte und Tests entstanden ist und welche verschiedenen Möglichkeiten vorhanden wären. Im zweiten Teil wird das Design der Benutzeroberfläche vorgestellt.

2 Analyse verwandter Produkte

2.1 Produkte

RTzen RT/shader

RT/shader wurde von der Firma RTzen im Sommer 2004 auf dem Markt eingeführt. RT/shader erlaubte als erstes Produkt das grafische Editieren von Shadern, wodurch Entwicklungszeiten und -kosten gesenkt werden konnten. Dadurch, dass das Tool auch von den Grafikern eingesetzt werden konnte wurde ebenfalls die visuelle Qualität der damit erstellten Produkte verbessert. Im Sommer 2005 wurde RTzen, kurz nach der Einführung der Version 2 von RT/shader, aufgekauft und RT/shader vom Markt genommen. Dadurch entstand die Marktlücke welche mit shaded gefüllt werden soll.

Nvidia FX Composer und ATI RenderMonkey

Diese beiden Tools wurden von den beiden Grafikkartenherstellern nVidia und ATI im Rahmen ihres jeweiligen Programms zur Unterstützung der Entwicklergemeinschaft herausgegeben. Die Tools können gratis von der Webseite des jeweiligen Herstellers heruntergeladen werden. Der Funktionsumfang ist bei beiden Produkten in ungefähr gleich gross, mit der Ausnahme von wenigen speziellen Features, wodurch je nach Anwendungsfall das eine oder das andere Produkt im Vorteil ist. Über ATI RenderMonkey kann unter [GDEV05] ein ausführliches Review gefunden werden.

Die folgenden Vor- und Nachteile können im Wesentlichen bereits in der Vision (siehe Kapitel I, Abschnitt 1) gefunden werden:

- Die Zielgruppe sind Software-Entwickler und nicht Grafiker. Für Grafiker ist die Verwendung eines der beiden Produkte schwierig und mit grossem Einarbeitungsaufwand verbunden.
- Auch wenn die Benutzeroberflächen teilweise Schwächen aufweisen, sind sie für Software-Entwickler einfach zu bedienen, da sie ähnlich wie die Oberfläche von Visual Studio.NET entworfen wurden. Auch für Nicht-Entwickler ist es von Vorteil, dass die Benutzeroberfläche durch das Docking an die individuellen Bedürfnisse des Benutzers angepasst werden kann.

Modelling Packages

Modelling Packages sind die grossen Softwarepakete mit welchen die Grafiker modellieren. Zur Zeit dieser Arbeit werden die folgenden Produkte verwendet:

Produkt	Beschreibung	Quellen
Autodesk 3ds Max	3ds Max ist bereits lange auf dem Markt. Das Studio ist die am weitesten verbreitete Software unter Spieleentwicklern. Der Hersteller Autodesk versucht diese Position mit einer agressiven Weiterentwicklung der Software zu behalten.	[WIKI02] [GDEV01] [GDEV02]
Autodesk Maya	Maya ist zur Zeit die am meisten verbreitete Software im Bereich der Filmindustrie. Zwei der Gründe dafür sind, dass einerseits die Benutzeroberfläche von Maya sehr leicht angepasst, bzw. sogar komplett ausgetauscht werden kann, und andererseits	[WIKI04] [GDEV04]

Produkt	Beschreibung	Quellen
	der RenderMan Standard von Pixar unterstützt wird.	
Maxon Cinema 4D	Cinema4D wird von der deutschen Firma Maxon produziert. Es ist vor allem beliebt, da es sehr leicht zu bedienen ist und eine sehr flexible Benutzeroberfläche besitzt.	[WIKI03] [GDEV03]
Blender	Blender ist frei verfügbar unter der General Public License. Dieses freie Softwarepaket bietet, wie die anderen drei kommerziellen Produkte, viele Features. Die Benutzeroberfläche gilt allgemein als innovativ und sehr flexibel, trotzdem hat Blender den Ruf, dass die Bedienung schwierig zu erlernen ist.	[WIKI10]

2.2 Erkenntnisse

Es konnten die folgenden für den Entwurf der Benutzeroberfläche von shaded relevanten Informationen gewonnen werden.

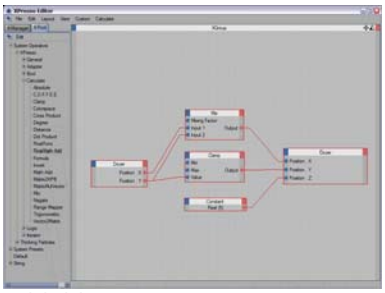


Abbildung 23: Xpresso, Skripteditor von Cinema 4D

- Die Modelling Packages verwenden teilweise ebenfalls Graphenstrukturen zur Darstellung. Als Beispiel dient der in Abbildung 23 gezeigte Script-Editor von Cinema 4D. Das heisst, dass Grafiker vielleicht schon an die Verwendung einer solchen Struktur gewöhnt sind.
- Praktisch alle Modellierungspakte besitzen zwei Symbolleisten. Jeweils eine am oberen und eine am linken Rand der Benutzeroberfläche. Die Verwendung der Symbolleisten ist hingegen unterschiedlich.
- Struktur- und Eigenschaftsansichten befinden sich bei praktisch allen Modelling Packages auf der rechten Seite der Benutzeroberfläche.
- Andockbare Fenster, bzw. Ansichten, sind bei allen untersuchten Shader-Editoren Standard. Bei den Modelling Packages sind andockbare Fenster unterschiedlich stark unterstützt. Generell wird die durch andockbare Fenster gewonnene Flexibilität geschätzt.

3 Entwurf

In Abbildung 24 wird der Entwurf der Benutzeroberfläche von shaded gezeigt. Darauf können die folgenden Elemente identifiziert werden:

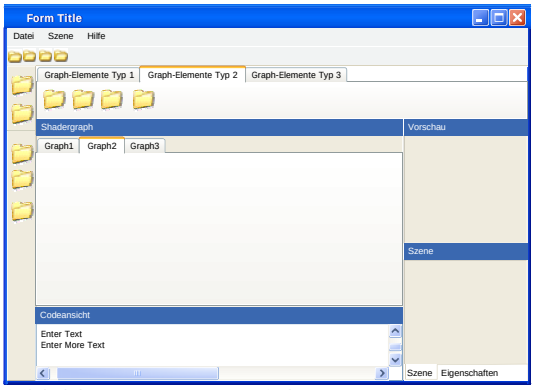


Abbildung 24: Entwurf der Benutzeroberfläche

3.1 Symbolleiste am oberen Rand

Die Symbolleiste am oberen Rand steuert die gleichen Dinge wie das Menu „Datei“ und ist daher identisch aufgebaut. Die einzige Ausnahme davon ist der Eintrag „Beenden“.

3.2 Symbolleiste am linken Rand

Die Symbolleiste am linken Rand enthält die gleichen Elemente wie das Menu „Szene“.

3.3 Menu

Das Menu befindet sich am oberen Rand. Es enthält die folgenden Menupunkte:

Datei

Dieses Menu enthält alle Einträge welche einen Zusammenhang mit Dateien haben. Es enthält daher in der folgenden Tabelle aufgeführten Einträge:

Eintrag	Beschreibung
Material erstellen	Über diesen Menueintrag kann der UC1 („Neues Material erstellen“) gestartet werden.
Material laden	Über diesen Menueintrag kann der UC2 („Material laden“) gestartet werden.
Material speichern	Über diesen Menueintrag kann der UC3 („Material speichern“) gestartet werden.
Material exportieren	Über diesen Menueintrag kann der UC4 („Material exportieren“) gestartet werden.
Modell importieren	Über diesen Menueintrag kann der UC6 („Modell importieren“) gestartet werden.
Szene laden	Über diesen Menueintrag kann der UC8 („Szene laden“) gestartet werden.
Szene speichern	Über diesen Menueintrag kann der UC9 („Szene speichern“) gestartet werden.
Szene zurücksetzen	Über diesen Menueintrag kann der UC10 („Szene zurücksetzen“) gestartet werden.
Beenden	Über diesen Menueintrag kann der Benutzer shaded verlassen.

Szene

Dieses Menu enthält alle Einträge über welche der Benutzer die aktuelle Szene modifizieren, respektive ein Werkzeug zur Modifikation der Szene auswählen kann. Es enthält die in der folgenden Tabelle aufgeführten Einträge:

Eintrag	Beschreibung
Ansicht rotieren	Über diesen Menueintrag wird der Szene-Rotationsmodus in der aktiven Vorschau-Ansicht aktiviert. Dadurch kann der Benutzer den UC13 („Ansicht auf Szene ändern“) starten.
Ansicht verschieben	Über diesen Menueintrag wird der Szene-Verschiebungsmodus in der aktiven Vorschau-Ansicht aktiviert. Dadurch kann der Benutzer den UC13 („Ansicht auf Szene ändern“) starten.
Ansicht vergrössern/verkleinern	Über diesen Menueintrag wird der Szene-Vergrösserungs/Verkleinerungs-Modus in der aktiven Vorschau-Ansicht aktiviert. Dadurch kann der Benutzer den UC13 („Ansicht auf Szene ändern“) starten.
Objekt rotieren	Über diesen Menueintrag wird der Objekt-Rotationsmodus in der aktiven Vorschau-Ansicht aktiviert. Dadurch kann der Benutzer, je nach dem ob eine Lichtquelle oder ein Modell selektiert ist, den UC11 („Modell positionieren“) oder den UC16 („Parameter einer Lichtquelle ändern“) starten.
Objekt verschieben	Über diesen Menueintrag wird der Objekt-Verschiebemodus in der aktiven Vorschau-Ansicht aktiviert. Dadurch kann der Benutzer, je nach dem ob eine Lichtquelle oder ein Modell selektiert ist, den UC11 („Modell positionieren“) oder den UC16 („Parameter einer Lichtquelle ändern“) starten.
Objekt entfernen	Über diesen Menueintrag kann der Benutzer, je nach dem ob eine Lichtquelle oder ein Modell selektiert ist, den UC12 („Modell entfernen“) oder den UC17 („Lichtquelle entfernen“) starten.
Zylinder erstellen	Über diesen Menueintrag kann der Benutzer den UC7 („Standardgeometrie erstellen“) starten und damit einen neuen Zylinder in die Szene einfügen.
Ebene erstellen	Über diesen Menueintrag kann der Benutzer den UC7 („Standardgeometrie erstellen“) starten und damit eine neue Ebene in die Szene einfügen.
Quader erstellen	Über diesen Menueintrag kann der Benutzer den UC7 („Standardgeometrie erstellen“) starten und damit einen neuen Quader in die Szene einfügen.
Teapot erstellen	Über diesen Menueintrag kann der Benutzer den UC7 („Standardgeometrie erstellen“) starten und damit einen neuen Teapot in die Szene einfügen.
Punktlichtquelle einfügen	Über diesen Menueintrag kann der Benutzer den UC15 („Lichtquelle einfügen“) starten und eine neue Punktlichtquelle in die Szene einfügen.

Ansicht

Dieses Menu enthält Menueinträge zur Steuerung der Benutzeroberfläche. Über dieses Menu können Teile der Benutzeroberfläche ausgeblendet, respektive ausgeblendete Elemente wieder eingeblendet werden. Es enthält die folgenden Einträge:

Eintrag	Beschreibung
Codeansicht	Dieser Menueintrag besitzt ein Auswahlkästchen. Es zeigt jeweils an, ob das Codeansichts-Element sichtbar ist oder nicht. Wählt der Benutzer den Menueintrag wenn das Auswahlkästchen markiert ist, das heisst wenn die Codeansicht sichtbar ist, dann wird sie aus der Benutzeroberfläche entfernt. Wählt der Benutzer den Menueintrag, wenn das Auswahlkästchen nicht markiert ist, dann wird die Codeansicht wiederum in die Benutzeroberfläche eingefügt.
Eigenschaftenansicht	Dieser Menueintrag besitzt ein Auswahlkästchen. Es zeigt jeweils an, ob das Eingenschaftenansicht-Element sichtbar ist oder nicht. Wählt der Benutzer den Menueintrag wenn das Auswahlkästchen markiert ist, das heisst wenn die Eingenschaftenansicht sichtbar ist, dann wird sie aus der Benutzeroberfläche entfernt. Wählt der Benutzer den Menueintrag, wenn das Auswahlkästchen nicht markiert ist, dann wird die Eingenschaftenansicht wiederum in die Benutzeroberfläche eingefügt.
Shadergraph	Dieser Menueintrag besitzt ein Auswahlkästchen. Es zeigt jeweils an, ob das Shadergraph-Element sichtbar ist oder nicht. Wählt der Benutzer den Menueintrag wenn das Auswahlkästchen markiert ist, das heisst wenn das Shadergraph-Element sichtbar ist, dann wird sie aus der Benutzeroberfläche entfernt. Wählt der Benutzer den Menueintrag, wenn das Auswahlkästchen nicht markiert ist, dann wird das Shadergraph-Element wiederum in die Benutzeroberfläche eingefügt.
Szenenhierarchie	Dieser Menueintrag besitzt ein Auswahlkästchen. Es zeigt jeweils an, ob das Szenenhierarchie-Element sichtbar ist oder nicht. Wählt der Benutzer den Menueintrag wenn das Auswahlkästchen markiert ist, das heisst wenn die Szenenhierarchie sichtbar ist, dann wird sie aus der Benutzeroberfläche entfernt. Wählt der Benutzer den Menueintrag, wenn das Auswahlkästchen nicht markiert ist, dann wird die Szenenhierarchie wiederum in die Benutzeroberfläche eingefügt.
Vorschau hinzufügen	Wenn der Benutzer auf diesen Menueintrag klickt, wird eine weitere Vorschau in die Benutzeroberfläche eingefügt.
Ansicht wiederherstellen	Wenn der Benutzer diesen Menueintrag wählt, wird die gesamte Benutzeroberfläche auf den Zustand zurückgesetzt, den sie beim Programmstart hatte.

Hilfe

Dieses Menu enthält die Hilfethemen. Über Hilfethemen kann der Benutzer Informationen über das Programm und dessen Anwendung erhalten. Die folgende Tabelle gibt einen Überblick über die Einträge in diesem Menu:

Eintrag	Beschreibung
Info	Öffnet einen Dialog mit Informationen über shaded. Die Informationen beinhalten im wesentlichen Autor und Version.

3.4 Vorschau-Ansicht

Die Vorschau-Ansicht befindet sich im Entwurf oben rechts. In ihr wird eine Vorschau der aktuellen Szene gezeigt, so dass der Benutzer das Resultat von Änderungen an Materialien und Szene sofort sehen kann. Optionen zur Navigation in der Szene befinden sich in einer Symbolleiste, welche sich direkt im dockbaren Fenster der Vorschau-Ansicht befindet. Die Vorschau-Ansicht kann später mehrmals in die Benutzeroberfläche eingefügt werden, so dass der Benutzer die Szene gleichzeitig aus verschiedenen Blickwinkeln betrachten kann, wenn er es möchte.

3.5 Eigenschaftenansicht

Auf der rechten unteren Seite der Benutzeroberfläche befinden sich zwei übereinander gedockte Fenster. Die Eigenschaftenansicht ist eines der beiden. In ihr werden die Eigenschaften des im Graph-Steuerelement angewählten Objektes angezeigt. Sie können auch in der Eigenschaftenansicht editiert werden.

3.6 Hierarchieansicht

Die Hierarchieansicht ist das zweite der übereinander gedockten Fenster auf der rechten unteren Seite der Benutzeroberfläche. In ihr werden mittels einer Baumstruktur die Zusammenhänge zwischen den einzelnen Objekten in der Szene dargestellt.

3.7 Codeansicht

Die Codeansicht zeigt jeweils den aus dem gewählten Shadergraph generierten und exportierten Shadercode.

3.8 Shadergraph

Das Graph-Steuerlement befindet sich im Zentrum der Benutzeroberfläche. Auf Ihm kann der Benutzer den Shadergraph eines Materials verändern. Es wird nach dem Vorbild von RT/shader beabsichtigt, dass die Zusammenhänge zwischen den Elementen einer Szene, das heisst vor allem zwischen Materialien und Modellelementen, ebenfalls in einer speziellen Art Graph modifiziert werden können. Damit verschiedene Graphen gleichzeitig geöffnet sein können, wird das Graph-Steuerelement in einem Registersteuerelement platziert, wobei für jeden geöffneten Shadergraphen ein eigenes Register zur Verfügung steht.

3.9 Tool-Registerelement

Das Tool-Registersteuerelement befindet sich in Abbildung 24 direkt unterhalb der Symbolleiste auf der oberen Seite des Fensters. Das Steuerelement präsentiert dem Benutzer die vom Shadergraphen unterstützen Elemente. Diese sollen dabei zur besseren Übersichtlichkeit in verschiedene Kategorien gruppiert werden. Über welchen Steuerelementtyp diese Ziele erreicht werden, soll über den UI-Prototypen evaluiert werden. Es bestehen die folgenden Varianten (Punkte a bis e) respektive Optionen (Punkt f) zur Positionierung und Verwendung dieses Elementes:

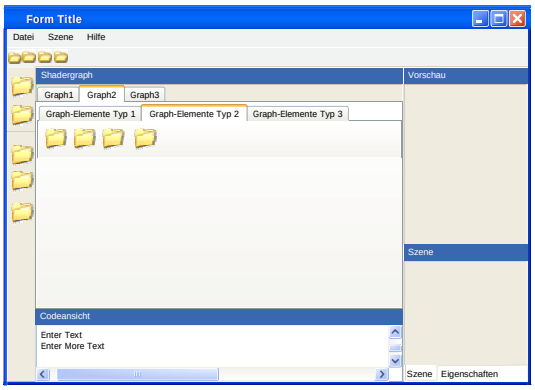


Abbildung 25: Alternative Benutzeroberfläche

k) Das Steuerelement wird wie in Abbildung 24 gezeigt unterhalb der horizontalen Symbolleiste und über dem Graph-Steuerelement angeordnet. Der Nachteil ist, dass das Steuerelement leer und ohne Zweck ist, solange kein Shadergraph geöffnet ist. Des weiteren müssen jedes Mal, wenn der Benutzer in das Register eines anderen Shadergraphen wechselt eventuell Inhalt und Struktur des Elementes verändert werden. Der Vorteil wäre, dass das Element an dieser Stelle jederzeit gut sicht- und erreichbar wäre.

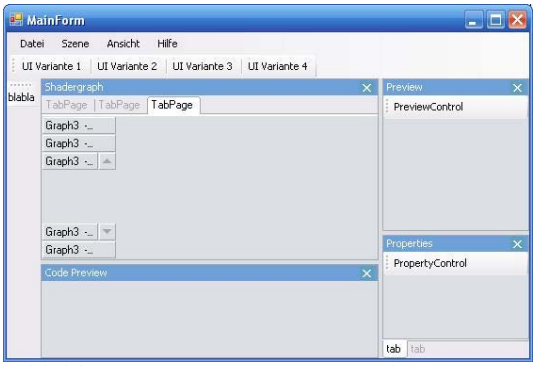


Abbildung 26: Variante c mit ToolBox-Steuerelement

l) Das Steuerelement wird wie in Abbildung 25 gezeigt als eine Art Symbolleiste (oder verschachteltes Register-steuerelement) innerhalb des Registers jedes Graphen geführt. Der Vorteil dieser Variante wäre, dass die Umstrukturierung bei einem Registerwechsel nicht mehr nötig wäre. Der Nachteil ist, dass es eventuell optisch nicht mehr so einfach zu finden ist und die Benutzbarkeit der Oberfläche darunter leiden könnte.

m) Diese Variante wurde von Simon Felix, einer Testperson, vorgeschlagen. Sie basiert darauf, dass das Tool-Registerelement so modifiziert wird, dass dessen Inhalt über ein anderes Steuerelement als das Registersteuerelement dargestellt werden kann. Die Inspiration dazu kam vom ToolBox-Steuerelement welches unter anderem in „3ds Max“, „Microsoft Visio“ oder „Microsoft Visual Studio.NET“ vorkommt. Die Idee war, dass diese Programme ähnliche Probleme zu lösen haben. Abbildung 26 zeigt wie der UI-Prototyp unter Verwendung dieses Vorschlages aussieht. Das ToolBox-Steuerelement besteht dabei aus

einer Reihe von Balken, welche die Kategorien darstellen. Diese Balken können aufgeklappt werden und zeigen dann die Elemente welche darin enthalten sind. Die Vorteile dieser Variante liegen auf der Hand. Einerseits kann das ToolBox-Steuerelement dort angeordnet werden, wo der Benutzer es erwartet und andererseits besteht das Problem der zwei verschachtelten Registersteuerelemente nicht mehr. Der Nachteil ist, dass damit ein weiteres Nicht-Standard-Steuerelement in die Benutzeroberfläche eingefügt wird, wodurch ein Mehraufwand für die Programmierung entsteht.

n) Diese Variante basiert ebenfalls auf einem Vorschlag von Simon Felix. Sie geht davon aus, dass zwei ineinander verschachtelte Registersteuerelemente dann stören, wenn sie direkt untereinander platziert sind. Die Variante soll Testen, ob dies auch der Fall ist, wenn sie zwar ineinander verschachtelt sind, aber durch Raum, bzw. ein weiteres Steuerelement, voneinander getrennt sind. Abbildung 27 zeigt einen Screenshot des UI-Prototypen unter Verwendung dieser Alternative. Eine Variation dieser Variante wäre auch die Register des Registersteuerelementes anders zu zeichnen, so dass das Registersteuerelement nicht mehr wie ein „typisches" Registersteuerelement aussieht, sondern wie etwas Neues aussieht. Dadurch könnte eventuell die Tatsache der zwei verschachtelten Registersteuerelemente vom Benutzer versteckt werden.

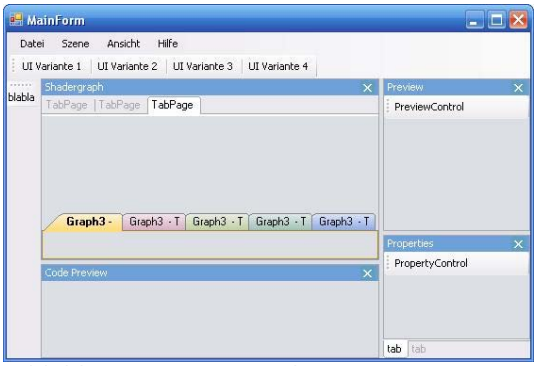


Abbildung 27: Variante d

o) Die letzte Variante wurde von Benjamin Sigg, einer weiteren Testperson, vorgeschlagen. Die Idee hinter dieser Variante ist, das Tool-Registersteuerelement auszublenden. Stattdessen soll eine Schaltfläche in die Benutzeroberfläche eingefügt werden. Diese würde dann so funktionieren, dass das Tool-Registersteuerelement für eine einmalige Verwendung eingeblendet wird, wenn der Benutzer den Mauszeiger über diese spezielle Schaltfläche bewegt oder darauf klickt. Diese Schaltfläche könnte dann ebenfalls innerhalb des Shadergraph-Elementes am logisch richtigen Ort hinterlegt werden. Der Vorteil dieser Variante ist wie bei allen Variationen, dass sich das Tool-Registersteuerelement am logisch richtigen Ort befindet. Der Nachteil ist, dass sich die Benutzer nicht daran gewöhnt sind, dass wichtige Elemente der Benutzeroberfläche ausgeblendet sind, und auf Wunsch für kurze Zeit eingeblendet werden können. Es besteht ebenfalls die Gefahr, dass die Benutzung des einblendbaren Elementes mühsam wird, wenn nicht sorgfältig getestet und ausbalanciert wird in welchen Momenten das Element ein- respektive ausgeblendet wird.

p) Es wäre eventuell lohnend den Inhalt der vertikalen Symbolleiste, welche Optionen zur Veränderung der Szene enthält, ebenfalls in ein Tool-Registerelement zu verpacken. Dieses Tool-Registerelement könnte angezeigt werden, wenn über das Graph-Steuerelement die Szene angezeigt wird. Dadurch könnte eventuell die vertikale Symbolleiste weggelassen und deren Funktionalität konsequenter eingebunden werden.

Welche der Varianten verwendet wird, soll über den UI Prototypen evaluiert werden können.

4 Design

Dieser Abschnitt beschreibt das Design der Benutzeroberfläche. Es wird jedoch nicht das Design des ersten Prototypen gezeigt, sondern dasjenige am Ende des Projektes.

Das Design des UI-Prototypen ist sehr einfach. In Abbildung 28 wird eine vereinfachte Version des Klassendiagrammes gezeigt.

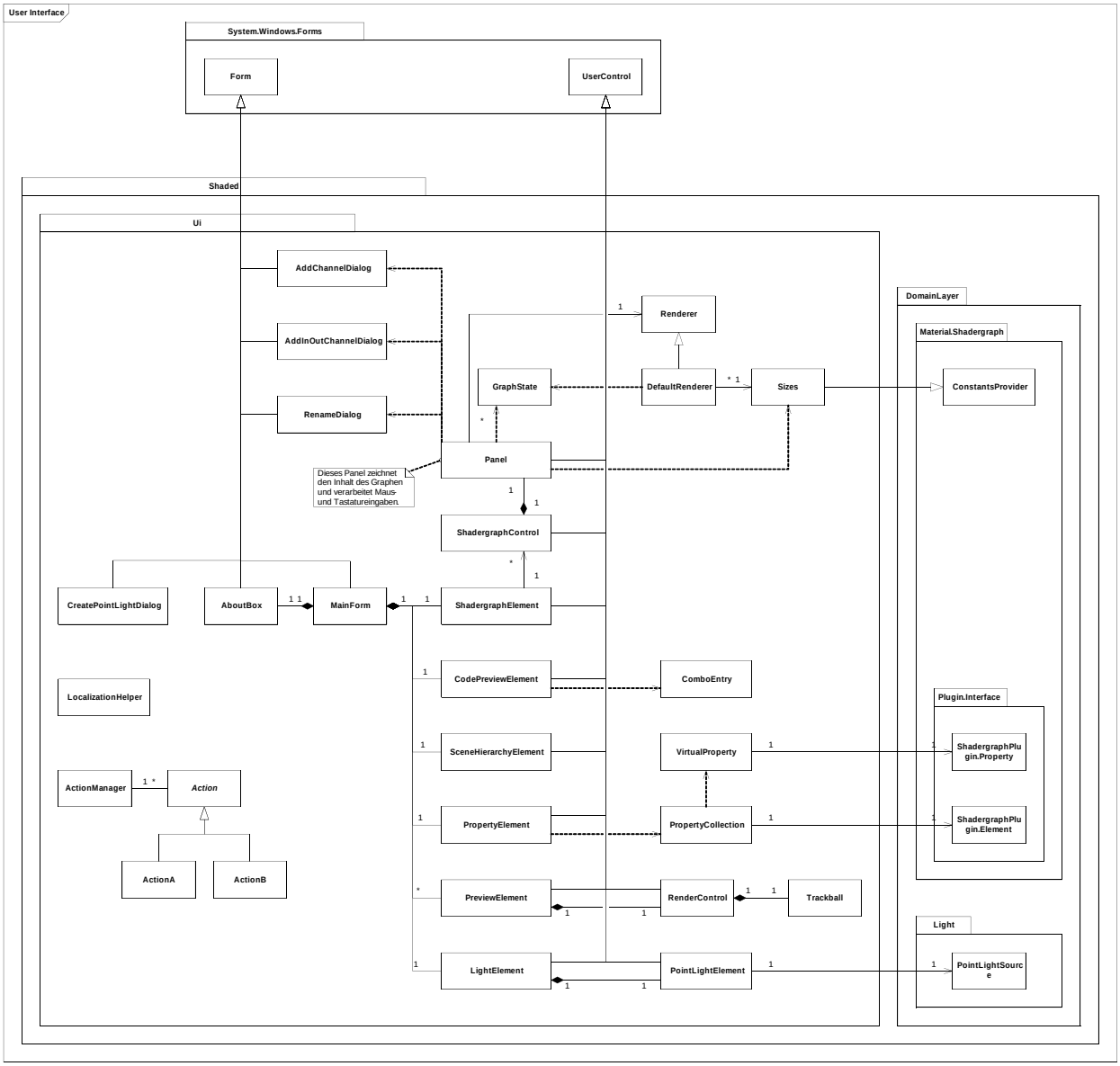


Abbildung 28: User Interface

Das wichtigste Konzept der Benutzeroberfläche ist die Klasse `ActionManager`. Sie verwaltet eine Reihe von Aktionen welche über die Benutzeroberfläche ausgelöst werden können. Dabei handelt es sich zum Beispiel um die Aktion zum Laden einer Szene, zum Importieren eines Modelles und Ähnlichem. Die vom `ActionManager` verwalteten Aktionen werden durch die Ereignissroutinen der Benutzeroberfläche aufgerufen. Die zugrunde liegende Idee ist, dass dadurch eine Aktion leicht von mehreren Orten von der Benutzeroberfläche aufgerufen werden kann und dadurch an einem zentralen Ort implementiert werden kann. In Abbildung 29 wird als Beispiel gezeigt welcher Code ausgeführt wird, nachdem der Benutzer im Menu gewählt hat, dass er ein neues Material erstellen möchte.

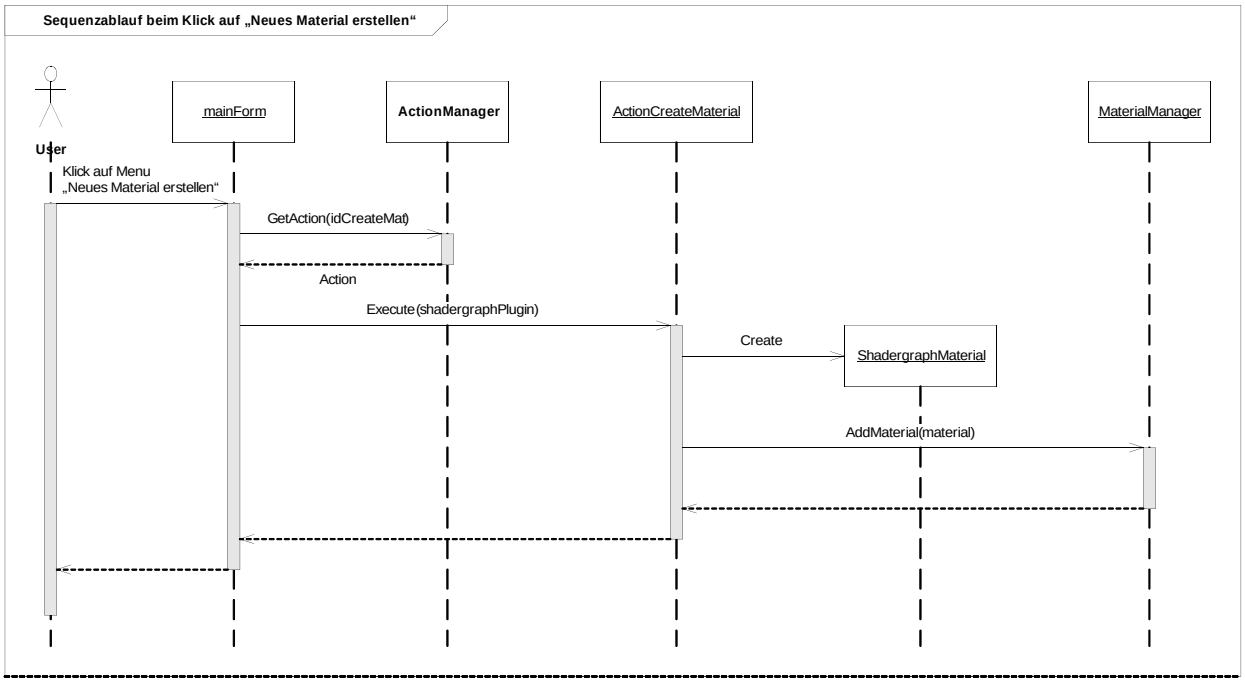


Abbildung 29: Typischer Ablauf unter Verwendung des ActionManagers

Kapitel VI: Scene

1 Einführung

Als Szene wird eine Anordnung von 3-Dimensionalen Objekten im Raum bezeichnet. Sie speichert sämtliche Informationen die benötigt werden, um diese Objekte manipulieren und rendern zu können. Dazu gehören Angaben über die Position, lokale Orientierung, Geometrie und verwendeten Materialien der Objekte.

Üblicherweise werden diese Informationen in einer Baumstruktur, einem so genannten Scene-Tree gespeichert. Diese Baumstruktur erlaubt es, die Objekte in Bezug auf die wichtigsten der oben genannten Attribute zu sortieren und dadurch sehr leicht manipulieren zu können.

Die Abbildung 30 zeigt als Beispiel eine Szene mit einem drehbaren Scheinwerfer. Der Scheinwerfer steht dabei auf einem Sockel. Wenn nun die dem Sockel übergeordnete Transformation geändert wird, wird der Scheinwerfer als komplette Einheit verschoben. Soll der Scheinwerfer jedoch nur auf dem Sockel rotiert werden, kann die dem Sockel unter- und dem Scheinwerfer übergeordnete Rotation verändert werden. Sobald diese Rotation verändert wurde, wird sowohl das Modell in der Szene anders gerendert werden, als auch der Scheinwerfer einen anderen Teil der Szene beleuchten.

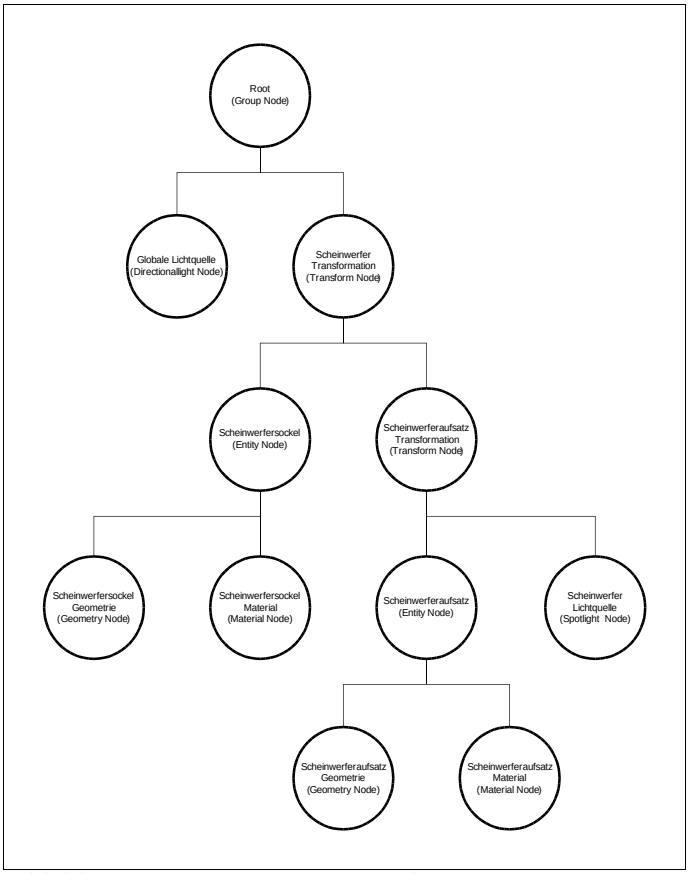


Abbildung 30: Scene-Tree eines Scheinwerfers

In shaded stellt die Szene den zentralen Punkt der Applikation dar und gibt allen Elementen des Domain-Layers einen Zusammenhang. Zu jedem beliebigen Zeitpunkt kann nur eine einzige Instanz der Klasse Scene jeweils aktiv sein. Diese Instanz wird von den verschiedenen Elementen der Benutzeroberfläche dargestellt und kann daraus editiert werden. Die Klasse enthält eine Referenz auf die Wurzel des Scene-Trees welcher in den Vorschau-Ansichten gerendert wird. Dieser Scene-Tree ist in shaded sehr einfach implementiert und enthält nur die einfachsten Konzepte wie Knoten zur Gruppierung und Transformation von Kindknoten, und Knoten über welche Meshes und Materialien referenziert werden können. Für solche Referenzierungsknoten, sogenannte EntityNodes, müssen Materialien und Modelle bereitgestellt werden. Zusätzlich werden für ein erfolgreiches Rendering Lichtquellen benötigt.

Diese drei Arten von Resourcen werden über drei Manager-Klassen, welche ebenfalls von der Scene-Klasse verwaltet werden, bereitgestellt. Die Resourcen und die dafür implementierten Konzepte werden in den folgenden Abschnitten vorgestellt. Zusätzlich folgt ein Abschnitt über die von EntityNodes und den Modellen benötigten Geometrie-Klassen. Sämtliche Klassendiagramme zu den vorgestellten Konzepten konnten aus Platzgründen nicht der Dokumentation beigelegt werden. Sie sind im Verzeichnis `„dokumentation/Drawings/TechBericht – Scene/“` auf der CD zu finden.

2 Lichtquellen

Die vom Benutzer editierten Shadergraphen enthalten normalerweise Elemente zur Berechnung der Reaktion des Materials auf eine Lichtquelle. Damit der Benutzer den Effekt der Lichtberechnungen im Shader in der Vorschau sehen kann muss er der Szene Lichtquellen hinzufügen. Diese Lichtquellen werden von einer Instanz der Klasse LightManager verwaltet. Jede der Lichtquellen in der Szene besitzt ein eindeutiges Identifikationsmerkmal über welches es im Shadergraph referenziert wird. Aus diesen Voraussetzungen folgt auch, dass eine Lichtquelle in der Szene erst dann den vollen Effekt besitzt, wenn sie von sämtlichen von ihr beleuchteten Materialien berücksichtigt, das heisst im Shadergraph integriert, wird.

3 Modelle

Wenn der Graphiker einen Shadergraphen editiert, so erstellt er ihn meist für ein bestimmtes Modell. Er möchte daher den Effekt des aus dem Graphen resultierenden Shaders vorzugsweise direkt am richtigen Modell sehen können. Diese Modelle können in shaded geladen werden und werden von einer Instanz der Klasse ModelManager verwaltet. Zusätzlich zu den aus Dateien geladenen Modellen besitzt shaded die Fähigkeit einige verschiedene geometrische Objekte prozedural zu generieren.

In Abbildung 31 wird die grobe Struktur der Modell-Klasse gezeigt. Sie zeigt, dass ein Modell aus einer Anzahl von Meshes besteht. Ein Mesh ist definiert als Ansammlung von Dreieckspolygonen welche alle mit demselben Material gerendert werden können. Diese Unterscheidung zwischen Modell und Mesh ist nötig, da ein Modell, zum Beispiel ein Auto mit Scheiben, Reifen und Karosserie, aus verschiedenen Teilen mit verschiedenen Materialien bestehen kann.

Ein Mesh wiederum besteht aus verschiedenen Streams von verschiedenen geometrischen Daten wie zum Beispiel Positionen, Normalvektoren oder Texturkoordinaten. Für einen Mesh muss zwingend ein Stream mit Positionen vorhanden sein. Alle anderen Streams Daten sind optional und nur vorhanden, falls sie in der Modelldatei vorhanden waren. Es werden jedoch nicht zwangsläufig immer alle Streams von einem Shader benötigt. Shaded lädt jedoch immer alle vorhandenen Streams, so dass immer auch die komplizierten Shader bedient werden können. Es ist definiert, dass die Indices in allen Streams aus derselben Anzahl von Polygonen bestehen müssen, und dass die Reihenfolge der Polygone in allen Streams dieselbe sein muss.

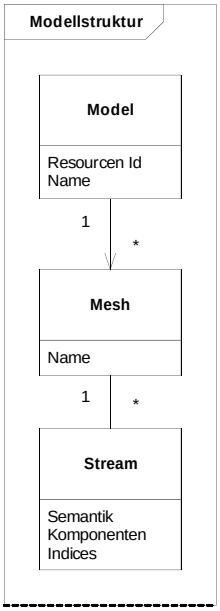


Abbildung 31: Modellstruktur

4 Geometrie

Die im letzten Abschnitt vorgestellte Art und Weise Modelldaten zu verwalten ist praktisch für das Laden und Bearbeiten der Modelle. Die Daten können in diesem Format jedoch nicht zum Rendering an die Grafikkarte übergeben werden. Dafür müssen die Modelldaten zuerst in ein anderes Format konvertiert werden. Dieses zur Grafikkarte kompatible Format wird im Geometry-Namespace umgesetzt.

Die Modelldaten werden dabei so umgewandelt, dass für jedes Polygon drei Vertices entstehen. Vertices sind Strukturen in denen alle Daten des Eckpunktes wie Position, Normalvektor, Texturkoordinaten und Ähnliches enthalten sind. Diese Vertices werden in einem Array in der Klasse ConcreteGeometry zusammen mit einem Indexarray gespeichert. Die folgende Tabelle zeigt als Beispiel die Eckpunkte einer Ebene:

		Position			Normalvektor			Texturkoordinaten	
Index	Polygon	X	Y	Z	X	Y	Z	U	V
0	1	-1	0	1	0	1	0	0	0
1	1	1	0	1	0	1	0	1	0
2	1	-1	0	-1	0	1	0	0	1
3	2	-1	0	-1	0	1	0	0	1

		Position			Normalvektor			Texturkoordinaten	
4	2	1	0	1	0	1	0	1	0
5	2	1	0	-1	0	1	0	1	1

5 Materialien

Nachdem Modelle und die daraus erstellten geometrischen Daten für die Grafikkarte vorgestellt wurden, wird in diesem Abschnitt das letzte für das Rendering von Objekten benötigte Konzept vorgestellt: die Materialien. Materialien werden von shaded dazu verwendet, um Shader zu kapseln. Dadurch sind sie primär verantwortlich für die Darstellung der Oberfläche von Objekten. Über den Vertexshader können Materialien Modelle zusätzlich in einem beschränkten Mass animieren. Materialien besitzen immer einen Vertex- und einen Pixelshader und alle für deren Verwendung benötigte Konstanten und Texturen. Ein Modell wird dann unter Verwendung dieser Attribute eines Materials gerendert.

Shaded verwendet die folgenden Materialien:

- Vordefinierte Materialien um Koordinatensysteme und Lichtquellen zu visualisieren.
- Ein Platzhaltermaterial welches auf alle Objekte angewendet wird welche vom Benutzer kein Material zugewiesen bekommen haben.
- Shadergraph-Materialien. Diese Materialien besitzen einen Shadergraphen im Hintergrund und verwenden das durch ihn generierte Shaderpaar. Dadurch implementieren die Shadergraph-Materialien die eigentliche Problemstellung und stellen somit eines der Kernstücke der Applikation dar.

Die Instanz der Klasse MaterialManager verwaltet sämtliche Shadergraph-Materialien und stellt die vordefinierten, intern verwendeten Materialien zur Verfügung.

Kapitel VII: Shadergraph Plugin

1 Übersicht

Das in diesem Kapitel vorgestellte Shadergraph Plugin wurde hauptsächlich in der zweiten Iteration implementiert. Die Entwicklung dieses Plugins stellte den riskanntesten Teil bei der Implementierung von shaded dar und umfasst im Wesentlichen die Anforderung FA6b. Das Shadergraph-Plugin implementiert ein Plugin über welches HLSL-Quellcode, passend für die Vertex- und Pixelshader-Version 3.0, generiert werden kann.

2 Design

2.1 Namespace-Übersicht

Das Konvertierungsplugin besteht grundsätzlich aus den folgenden vier Namespaces.

Namespace	Beschreibung
Shaded	Dieser Namespace repräsentiert die gesamte Applikation von shaded.
Elements	Der Elements-Namespace soll alle Klassen enthalten welche Teil des Shadergraphen sind. Dazu gehören unter anderem der Shadergraph selbst, die konkreten Elemente und Verbindungen. Im Wesentlichen befinden sich hier die Klassen, welche das Shadergraph-Plugin-Interface implementieren.
Hlsl	Der Hlsl-Namespace soll alle Elemente enthalten welche mit der High Level Shading Langauge zu tun haben. Dazu gehören die verschiedenen Objekte aus welchen ein Vertex- respektive Pixelshader zusammengesetzt ist.
Error	Der Error-Namespace soll die zur Fehlerbehandlung benötigten Elemente enthalten.
Test	Der Test-Namespace soll alle Klassen enthalten, welche zum Testen des Plugins benötigt werden. Bei der Implementierung wird dieser Namespace in ein separates Assembly ausgelagert.

Die Namespaces realisieren das in Abbildung 32 gezeigte Schichtenmodell. Der namespace `Test` wurde dabei weggelassen, da er ohne Rücksicht auf die Schichten auf alle beteiligten Namespaces und Klassen Zugriff hat.

Das Modell zeigt, dass vor allem der Namespace `Elements` das Plugin-Interface der Applikation implementiert. Diese wird danach über dieses Interface mit dem Plugin kommunizieren und den Graphen mit den im Namespace `Elements` vorhandenen Klassen aufbauen. Sobald die Applikation dem Plugin den Befehl gibt, den Graphen in ein Vertex- und Pixelshaderpaar umzuwandeln, wird der Namespace `Hlsl` benötigt. Zur Kompilation findet eine Kommunikation zwischen den beiden Namespaces `Elements` und `Hlsl` statt. Wenn Fehler auftreten so werden Exceptions aus dem Namespace `Error` geworfen. Diese Exceptions können danach in der Applikation gefangen und behandelt werden.

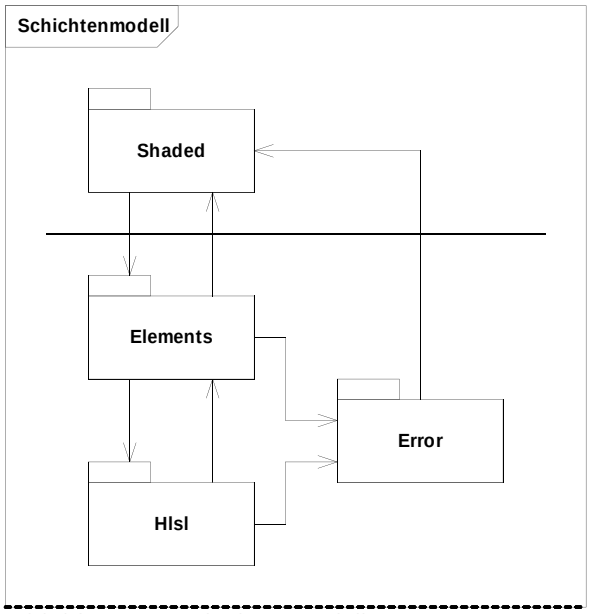


Abbildung 32: Schichtenmodell

2.2 Namespace Elements

Der Namespace `Elements` ist relativ gross und wird deshalb weiter in die folgenden sechs Bereiche unterteilt. In allen diesen Bereichen werden nur vereinfachte Klassendiagramme gezeigt. Das detaillierte würde ausgedruckt beinahe einen Quadratmeter in Anspruch nehmen und ist deshalb auf der CD Im Verzeichnis „`dokumentation/Drawings/TechBericht – Shadergraph Plugin`“ zu finden.

Interface ElementType und dazugehörende Klassen

Die Abbildung 33 zeigt die Klasse `ElementType` und alle dazugehörenden und abgeleiteten Klassen. Die Aufgabe der Klassen dieses Diagramms ist es, die von Element abgeleiteten Klassen zu beschreiben. Die in diesen Klassen enthaltenen Informationen sind:

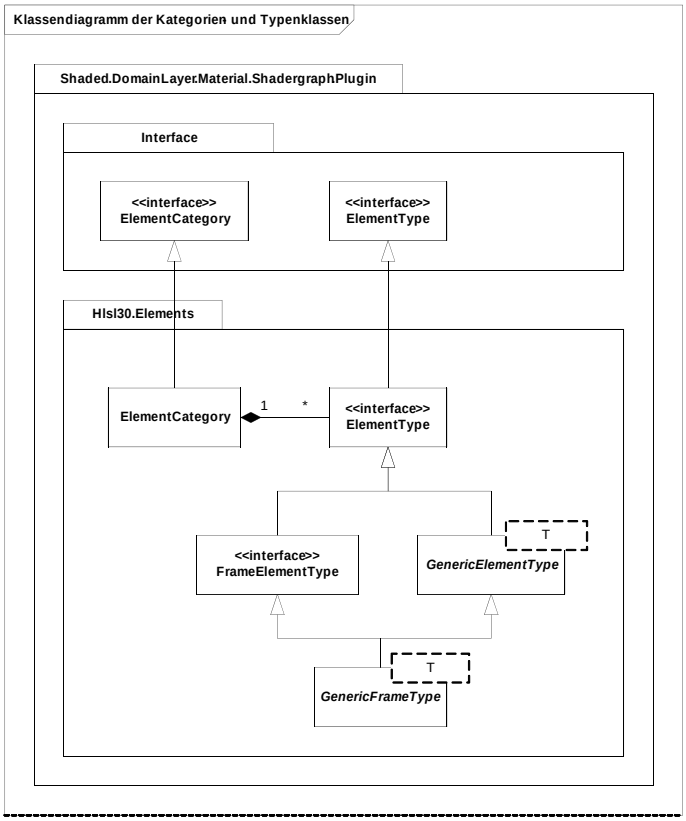


Abbildung 33: Klassendiagramm ElementType

- Der Name des Typs. Dieser Name wird für den Prototypen von shaded auf der Benutzeroberfläche angezeigt.
- Die auf der Benutzeroberfläche angezeigten Namen der Ein- und Ausgabekanäle.
- Die im generierten HLSL-Code verwendeten Variabelnamen der Ausgabekanäle.
- Die auf der Benutzeroberfläche angezeigten Namen der Eigenschaften eines Elementes.

Zusätzlich werden die von `ElementType` abgeleiteten Klassen Plugin-Intern als Factory-Klassen für die von ihnen beschriebenen von der Klasse `Element` abgeleiteten Klassen eingesetzt.

Interface Property

Die Abbildung 34 zeigt das Klassendiagramm der Klasse `Property` und der von ihr abgeleiteten Klassen. Über von Instanzen der Klasse `Property` speichern Elemente im Shadergraphen Eigenschaften, welche der Benutzer auf der Benutzeroberfläche verändern kann. Wenn Beispielsweise ein Element verschiedene Algorithmen unterstützt, so könnte der Benutzer über eine solche Eigenschaft wählen welchen Algorithmus er benutzen möchte.

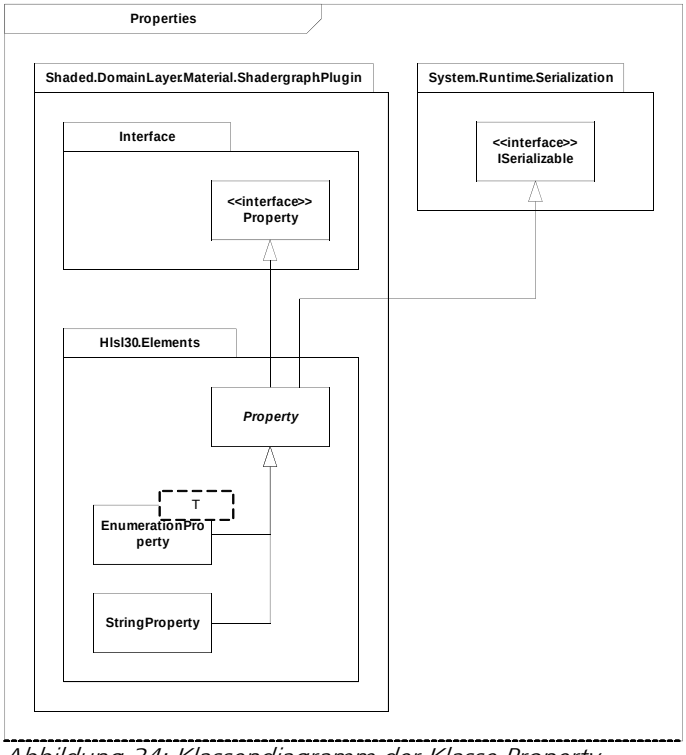


Abbildung 34: Klassendiagramm der Klasse Property

Klassen Input- und OutputChannel

In Abbildung 35 werden die Klassen für Ein- und Ausgabekanäle gezeigt. Elemente und Graphen verwenden Eingabekanäle dazu, um Daten von anderen Elementen zu erhalten und Ausgabekanäle um verarbeitete Daten an andere Elemente weiterzugeben.

Innerhalb des Plugins wird unterschieden zwischen Element-Kanälen und Graph-Kanälen. Die Element-Kanäle sind Elementen zugeordnet und werden von diesen Verwendet. Graph-Kanäle hingegen stellen die Ein- und Ausgabedaten des Shadergraphen oder eines Subgraphen bereit.

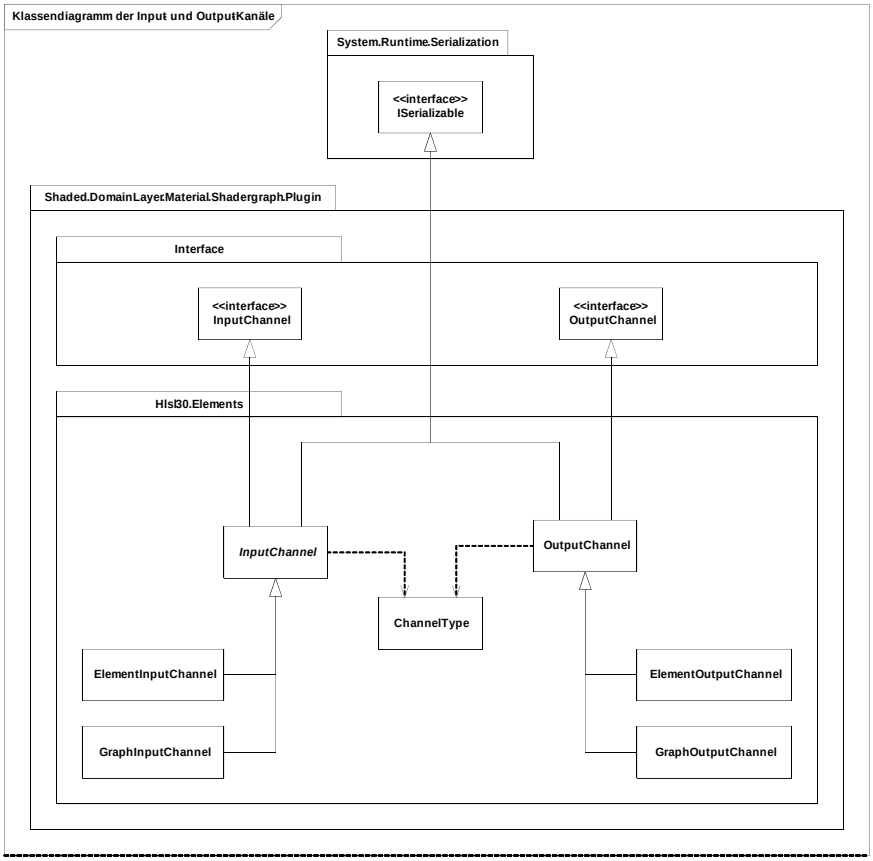


Abbildung 35: Klassendiagramm der Klassen InputChannel und OutputChannel

Klasse ChannelType

Die verschiedenen Elemente können nicht immer alle Datentypen verarbeiten. Statt dessen bestimmen sie für jeden ihrer Eingabekanäle, welche Datentypen sie über ihn entgegennehmen und verarbeiten können. Zusätzlich bestimmen sie für jeden Ausgabekanal ebenfalls, welche Datentypen über den Ausgabekanal ausgegeben werden können. Diese Angaben müssen sich nicht unbedingt auf einen einzelnen Typen beschränken. Statt dessen kann ein Kanal auch viele verschiedene Datentypen entgegennehmen. Zum Beispiel kann ein Kanal angeben, dass er einen Fliesskommazahl-Vektor von beliebiger Dimension verarbeiten kann. In HLSL besteht ein Datentyp grundsätzlich aus den folgenden Komponenten:

- Seinen eigentlichen Datentyp. Dadurch wird zwischen Fliesskommazahlen, integrale Zahlen, boolsche Werte und ähnlichen Typen unterschieden. Diese Angabe wird im Plugin auch TemplateTyp genannt.
- Seinen sogenannten. Komponententyp. Der Komponententyp bestimmt, ob es sich um einen skalaren Wert, einen Vektor oder eine Matrix handelt.
- Seine Dimensionen, falls es sich um Vektoren oder Matrizen handelt.

Um alle Kombinationen abzudecken, wurde die Klasse `ChannelType` entworfen. Diese Klasse kann über Bitfelder sämtliche verschiedenen möglichen Kombinationen in HLSL abdecken. Durch eine Instanz der Klasse `ChannelType` kann ein Ein- bzw. Ausgabekanal deklarieren welche Datentypen er verarbeiten oder ausgeben kann. Das wiederum vereinfachte Klassendiagramm in Abbildung 36 zeigt die Klasse `ChannelType` und die von ihr verwendeten Klassen.

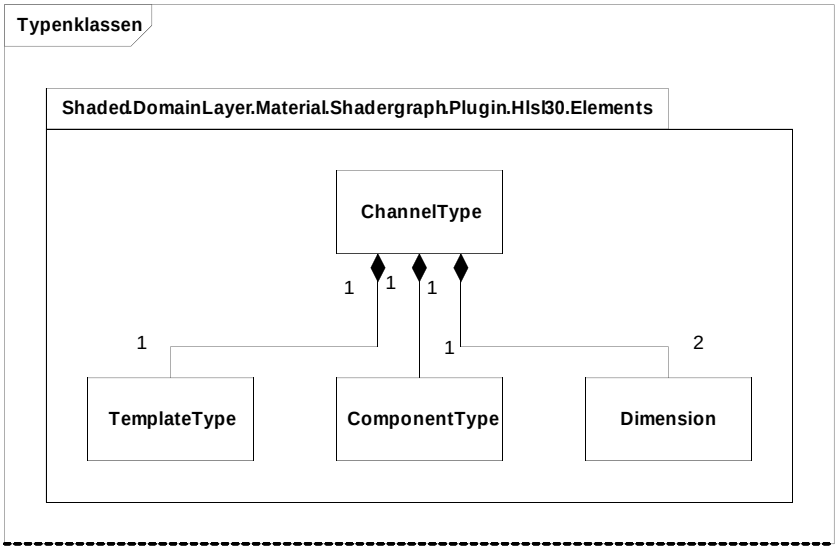


Abbildung 36: Klassendiagramm der Klasse ChannelType

Das Typenkonzept ist eines der kompliziertesten Konzepte im HLSL30 Plugin. Daher soll in diesem Abschnitt kurz näher darauf eingegangen werden. Insbesondere die sogenannten `Intrinsic-Elemente` des Graphen stellen hohe Anforderungen an die Flexibiltät des Typenkonzeptes. `Intrinsic-Elemente` sind diejenigen Elemente, welche in HLSL eingebaute Funktionen kapseln. Dazu gehören beispielsweise `mul`, `dot`, `cross`, `cos` und ähnliche Funktionen. Eine der aufwändigsten dieser Funktionen ist mit Sicherheit die Funktion `mul`. Diese Funktion soll daher als Beispiel dienen. Die Funktion `mul` repräsentiert die Multiplikation zwischen zwei Variabeln und besitzt daher zwei Eingabekanäle, für jeden Operanden einen, und einen Ausgabekanal für das Resultat der Multiplikation. Dadurch, dass HLSL Skalartypen, Vektortypen und Matrixtypen besitzt, wurde die Funktion der Multiplikation mehrfach überladen, wodurch diverse Kombinationen von Ein- und Ausgabetypen möglich sind. Die folgende Tabelle zeigt diese Kombinationen:

Nummer	Linker Operand	Rechter Operand	Resultat
1	Skalar	Skalar	Skalar
2	Skalar	Vektor, Dimension n	Vektor, Dimension n
3	Skalar	Matrix, Dimension n x m	Matrix, Dimension n x m
4	Vektor, Dimension n	Skalar	Vektor, Dimension n
5	Vektor, Dimension n	Vektor, Dimension n	Skalar
6	Vektor, Dimension n	Matrix, Dimension n x m	Vektor, Dimension m
7	Matrix, Dimension n x m	Skalar	Matrix, Dimension n x m
8	Matrix, Dimension n x m	Vektor, Dimension m	Vektor, Dimension n
9	Matrix, Dimension n x m	Matrix, Dimension m x k	Matrix, Dimension n x k

Dadurch, dass die Funktion vielfach überladen ist, entsteht eine komplizierte Wechselwirkung mit anderen, verbundenen Elementen. Wenn beispielsweise der erste Eingabekanal, das heisst der erste Operand, mit einem Ausgabekanal eines anderen Elementes verbunden wird, können Rückschlüsse gezogen werden, welche der überladenen Funktionen noch möglich sind. Zum Beispiel könnte der Ausgabekanal spezifizieren, dass er Fliesskommazahl-Vektoren der Dimension drei ausgibt. Durch diese Angabe kann das Multiplikationselemente alle Überladungen bis auf die Nummern 4, 5 und 6 auszuschliessen. Weil nur diese drei Überladungen in Frage kommen, und die Dimension des Vektors auf dem ersten Eingabekanal bestimmt ist, können die Kanaltypen auf dem zweiten Eingabekanal und dem Ausgabekanal ebenfalls eingeschränkt werden.

Daraus ist gut ersichtlich, dass ein Ausgabekanal den Datentypen nicht immer genau spezifiziert. Stattdessen kann er auch eine beliebige Kombination von Typen ausgeben und diese Kombination später, basierend auf

genaueren Angaben bei den Eingabekanälen weiter einschränken. Diese Einschränkung der Ausgabe muss dann jedoch an alle mit dem Ausgabekanal verbundenen Elemente weitergeleitet werden, so dass diese ebenfalls ihre Ein- und Ausgabetypen darauf basierend genauer spezifizieren können. Durch dieses immer genauere Spezifizieren der Datentypen ist es möglich, dass nach dem Erstellen einer neuen Verbindung der ganze Graph mehrmals durchlaufen werden muss, bis alle Elemente wieder korrekte Angaben zu ihren Ein- und Ausgabetypen berechnet haben.

Interface Element

Die Eigenschaften und Methoden von Elementen wurden im HLSL30-Plugin so weit als möglich abstrahiert und in allgemeinen Klassen zusammengefasst. Die Klassen bieten eine allgemeine Implementierung der meisten Funktionen eines Elementes. Um ein Element zu implementieren muss von einer der generellen Klassen abgeleitet werden. Danach kann die gewünschte Funktionalität durch Implementieren und Überschreiben der geeigneten Methoden erreicht werden. Die Abbildung 37 zeigt das Klassendiagramm dieser generischen Klassen.

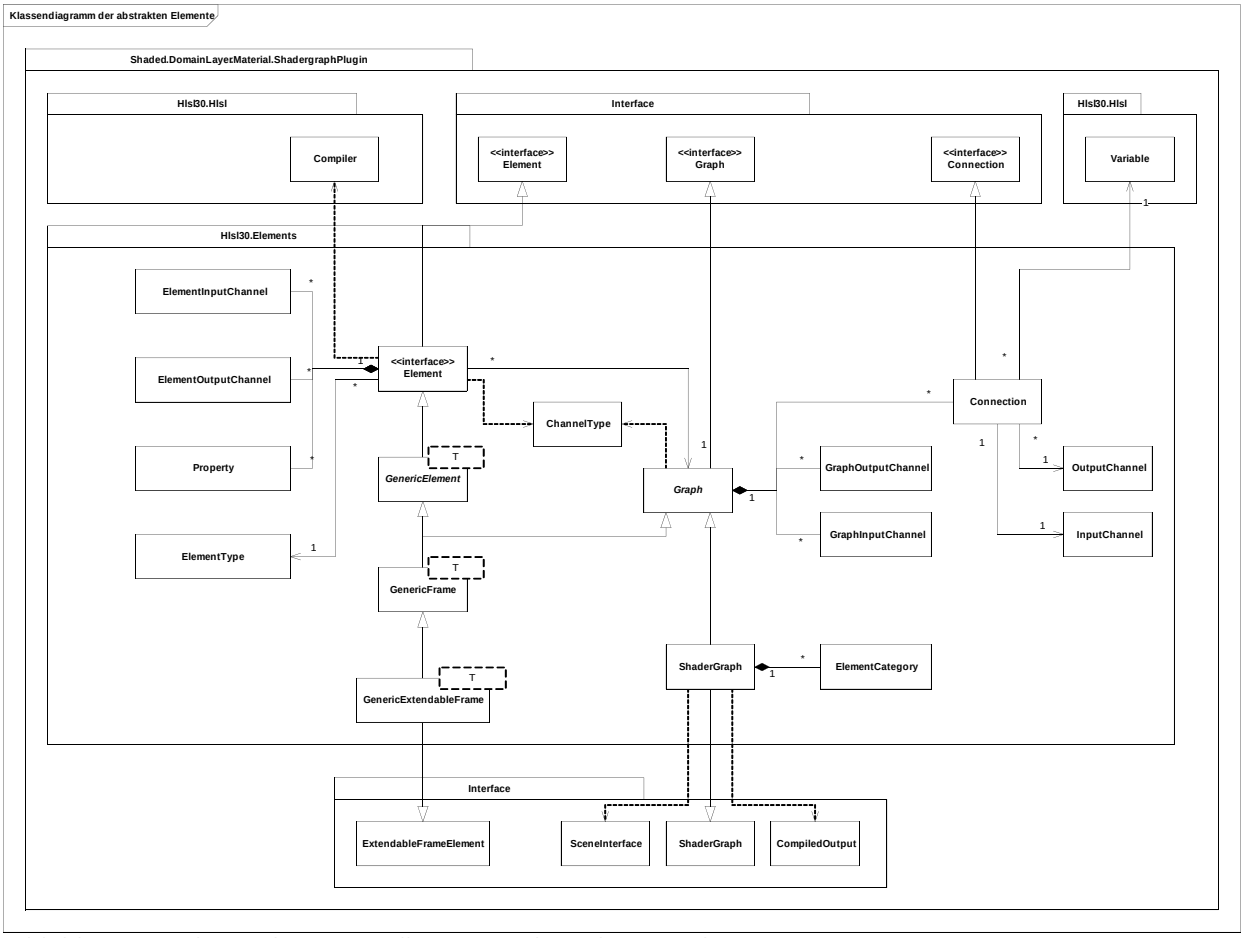


Abbildung 37: Klassendiagramm aller generischen Element-Klassen

2.3 Namespace Hlsl

Der Namespace `Hlsl` umfasst sämtliche Klassen, welche Sprachkonstrukte von HLSL abstrahieren. Die Klassen und Interfaces dieses Namespace wurden zur besseren Übersicht in den Klassendiagrammen ebenfalls wiederum in mehrere Teile aufgeteilt. Im Folgenden werden wie im letzten Abschnitt nur vereinfachte Klassendiagramme gezeigt. Das detaillierte Klassendiagramm kann auf der CD im Verzeichnis **„`dokumentation/Drawings/TechBericht – Shadergraph Plugin`“** gefunden werden.

Klassen Compiler und Shader

Die zentrale Klasse für das Umwandeln des Shadergraphen in HLSL-Quellcode ist die Klasse `Compiler`. Diese Klasse speichert die Resultate während dem Kompiliervorgang. Sie enthält hauptsächlich jeweils eine Instanz der Klassen VertexShader und PixelShader. Danach gibt der Compiler den Elementen, sobald sie bereit sind, die Gelegenheit HLSL-Strukturen zu bilden. Die Abbildung 38 zeigt ein vereinfachtes Klassendiagramm der Klassen `Compiler`, `Shader`, `PixelShader` und `VertexShader` und aller damit verbundenen Klassen.

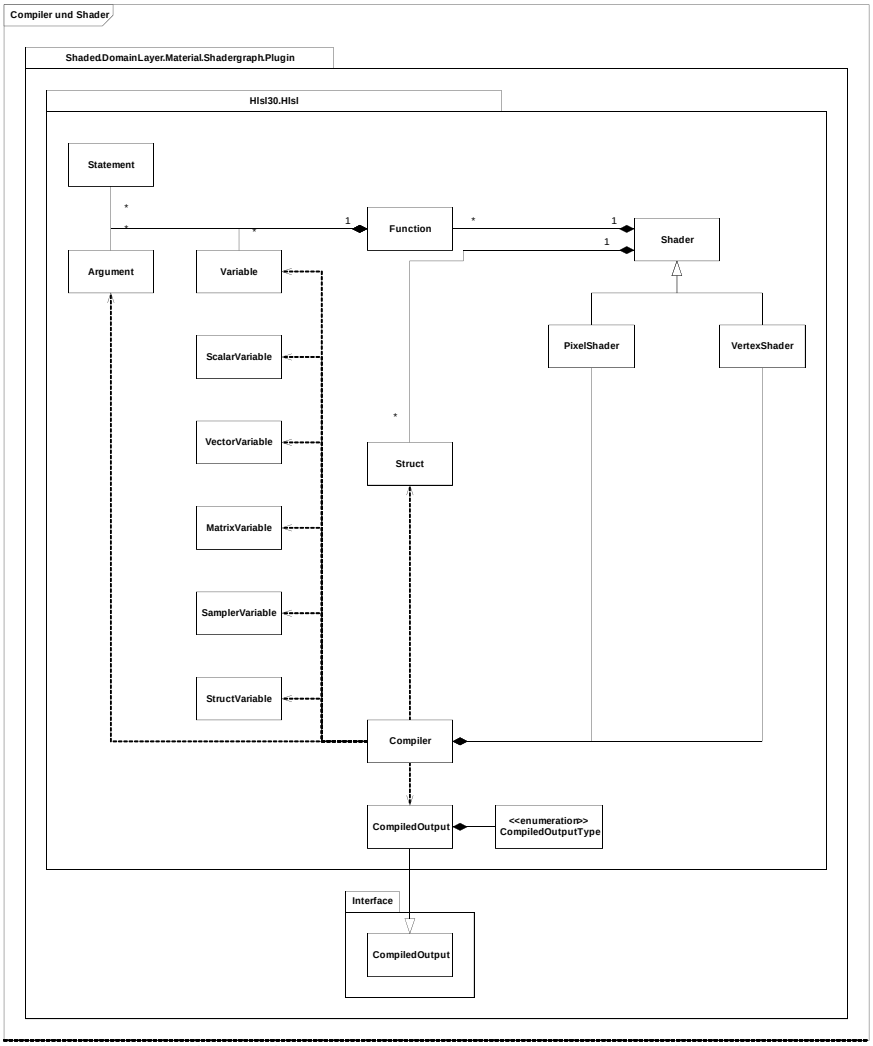


Abbildung 38: Klassendiagramm der Klassen Compiler und Shader

Der im Codeabschnitt 2 gezeigte Algorithmus zur Generierung des Quellcodes ist relativ einfach. Es gilt die Regel, dass ein Element Code generieren kann, sobald es auf allen Eingabekanälen Daten, das heisst eine Variable, anliegen hat. Diese Überprüfung und eventuelle spezielle Überprüfungen führen die Elemente selber aus. Code wird generiert indem alle Elemente des Shadergraphen in einen Container kopiert werden. Danach wird über alle Elemente des Containers iteriert. Wenn der Compiler ein Element findet, das bereit ist Code zu generieren, so wird ihm die Möglichkeit gegeben Code zu generieren. Nachdem es Code generiert hat wird es aus dem Container entfernt und der Compiler beginnt wieder von vorne. Es wäre möglich, dass eine Unendlichschleife entsteht, wenn ein Fehler auftritt der zur Folge hat, dass danach kein Element der Liste mehr Code generieren kann. Deshalb wird eine entsprechende Überprüfung vorgenommen.

Frames haben ebenfalls die Aufgabe, Code aus einem Graphen zu generieren. Sie verwenden dazu wie der Compiler für den Shadergraphen den Algorithmus, welcher im Codeabschnitt 2 gezeigt wird.

```
todo = new List(graph.Elements)
breaked = false
while (todo.Count > 0)
{
  for (i = 0; i < todo.Count; ++i)
  {
    element = todo[i];
    if (element.CanGenerateCode())
    {
      element.GenerateCode(this);
      todo.Remove(element);
      breaked = true;
      break;
    }
  }
  if (todo.Count > 0 && !breaked)
  {
    foreach (ShaderGraphElements.Element element in todo)
    {
      element.ReportProblems()
    }
    throw new Error.InfiniteLoopException()
  }
  breaked = false
}
```

Codeabschnitt 2: Algorithmus zur Codegenerierung (Vereinfachter C#-Code)

Klasse Variable

Innerhalb des HLSL30-Plugins wird für jeden Ausgabekanal eine Variable alloziert. Diese Variable speichert das Resultat welches über den mit ihr assoziierten Ausgabekanal an andere Elemente weitergegeben wird. Es gilt die Regel, dass eine Variable nur einmal gesetzt werden darf, jedoch beliebig oft gelesen werden kann. Durch diese Regel kann der Quellcode später ohne Probleme anders geordnet werden, solange darauf geachtet wird, dass alle Variablen immer zuerst initialisiert werden bevor sie genutzt werden. Für die Übertragung von verschiedenen Datenwerten werden verschiedene Typen von Variablen benötigt. Die Abbildung 39 zeigt das vereinfachte Klassendiagramm der Variablen.

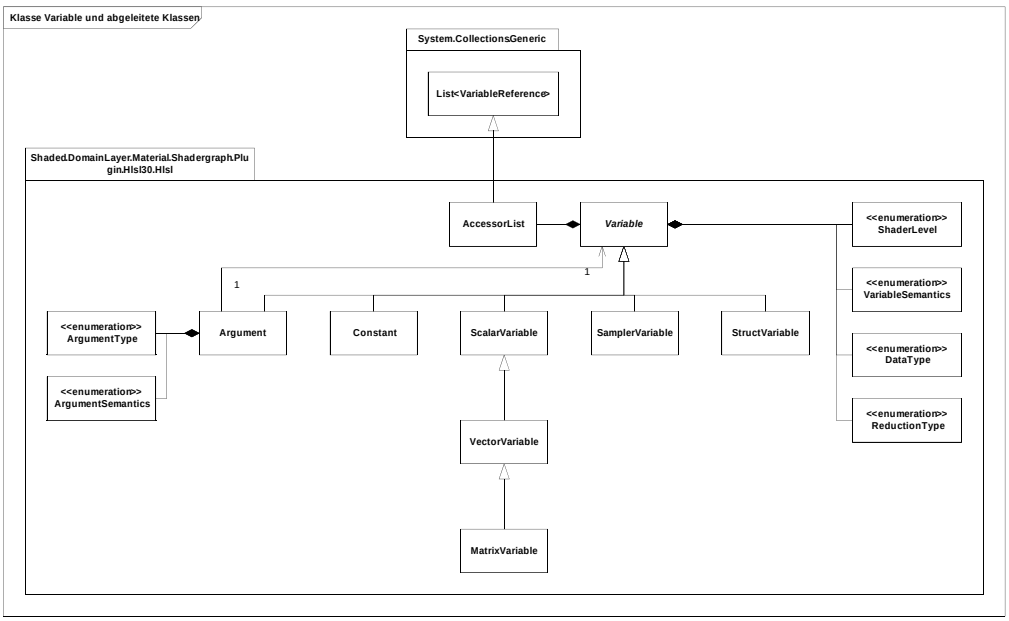


Abbildung 39: Klassendiagramm der Klasse Variable

Interface Statement

Die Elemente des HLSL30-Plugins generieren nicht direkt Quellcode. Stattdessen konstruieren sie Strukturen aus Instanzen der von `Statement` abgeleiteten Klassen. Diese internen Strukturen haben den Vorteil, dass sie leicht optimiert werden können, so dass der generierte Shader durch den Benutzer leicht lesbar wird. Zudem stellen sie Methoden zur Verfügung, mit welchen die Strukturen leicht in HLSL Quellcode umgewandelt werden können. Die Abbildung 40 zeigt das vereinfachte Klassendiagramm der Klasse `Statement` und aller von ihr abgeleiteten Klassen.

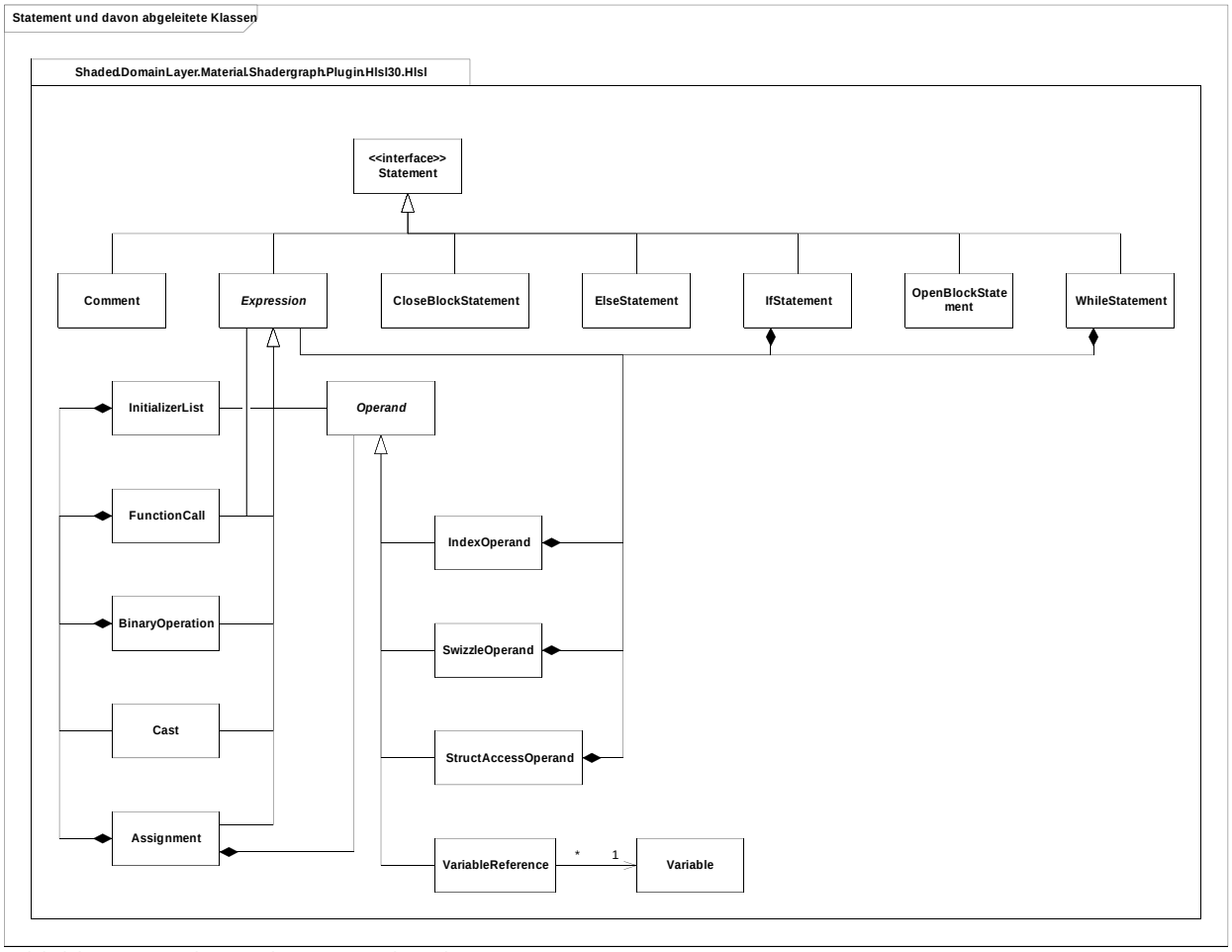


Abbildung 40: Klassendiagramm der Klasse Statement

2.4 Namespace Error

Der Namespace Error enthält die fünf Exceptions, welche im Fehlerfall vom Plugin geworfen werden. Diese Exceptions sind:

- `InfiniteLoopException`: Diese Exception wird entweder von einem Frame oder dem Compiler geworfen, wenn ein Fehler aufgetreten ist, der verhindert dass Elemente Code generieren können, wodurch eine Unendlichschlaufe beim Kompilieren entstehen würde.
- `InvalidConnectionException`: Diese Exception wird geworfen, wenn Client-Code versucht eine Verbindung zwischen zwei nicht kompatiblen Kanälen herzustellen.
- `InvalidOperationException`: Diese Exception wird geworfen, wenn von Client-Code eine Methode oder Property auf einem Objekt aufgerufen wurde, die nicht aufgerufen werden darf.
- `InvalidParameterException`: Diese Exception wird von einem Konstruktor oder einer

Methode geworfen, wenn ihr ein ungültier Parameter übergeben worden ist.

- `NotYetSupportedException`: Diese Methode wird geworfen, wenn eine Operation ausgeführt werden soll, die in dieser Version der Software noch nicht unterstützt wird.

Kapitel VIII: Umsetzung

1 Assemblies

Die fertige Software besteht aus den folgenden neun Assemblies:

Assembly	Beschreibung
Kaos.Ui.dll	Dieses Assembly enthält ein verbessertes TabView-Steuerelement und ein Steuerelement für andockbare Fenster. Diese beiden Steuerelemente wurden nicht im Rahmen der Studienarbeit entwickelt.
Shaded.exe	Das .exe-Assembly enthält den gesamten Code der Benutzeroberfläche.
Shaded.Core.dll	Dieses Assembly enthält alle im Kapitel IV vorgestellten Klassen des shaded Kerns.
Shaded.DomainLayer.dll	Dieses Assembly enthält alle Klassen des DomainLayers mit Ausnahme der Plugins. Zu den Klassen des DomainLayers gehören alle Klassen der Szene, die Plugin-Interfaces und der PluginManager.
Shaded.DomainLayer.Material.Shadergraph.ExportPlugin.Fx.dll	Dieses Assembly enthält ein Export-Plugin. Dieses Export-Plugin kann Shader des HLSL30-Shadergraph-Plugins in das .fx-Dateiformat von DirectX exportieren.
Shaded.DomainLayer.Material.ShaderGraph.Plugin.Hlsl30.dll	Dieses Assembly enthält das HLSL30-Shadergraph-Plugin von shaded.
Shaded.DomainLayer.Material.ShaderGraph.Plugin.Hlsl30.Test.exe	Dieses Assembly enthält sämtliche Tests für das HLSL30-Shadergraph-Plugin. Dazu gehören die Unit-Tests und die Konsolentestapplikation.
Shaded.DomainLayer.Serialization.ASE.dll	Dieses Assembly enthält das .ASE.Modelimport-Plugin von shaded.
ToolBox.dll	Dieses Assembly enthält ein ToolBox-Steuerelement. Das Steuerelement stammt von [COPR01].

2 Screenshots der Applikation

Die folgenden Bilder sollen einen kurzen Eindruck der Applikation geben:

- Abbildung 41: Diese Abbildung zeigt die Shadergraph-Ansicht von shaded. Dieses Steuerelement dient dazu, ihn zu verändern.
- Abbildung 42: Die Abbildung zeigt die Code-Ansicht von shaded. In ihr wird der aus dem Shadergraphen generierte Quellcode dargestellt. Der Benutzer kann so im voraus erkennen, wie der Shader exportiert werden wird.
- Abbildung 43: Die Abbildung zeigt rechts aussen die Vorschau-Ansicht von shaded. In der Vorschau-Ansicht sieht der Benutzer visuell was resultiert wenn der Shader auf ein

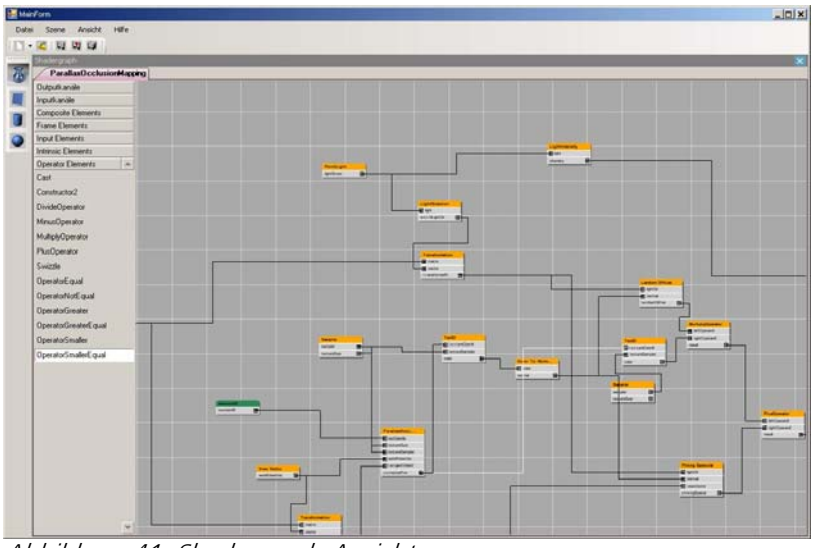


Abbildung 41: Shadergraph-Ansicht

Modell angewendet wird. In diesem Screenshot ist eine Implementation des „Parallax Occlusion Mapping“. Die Technik wird beschrieben in [NATA01]

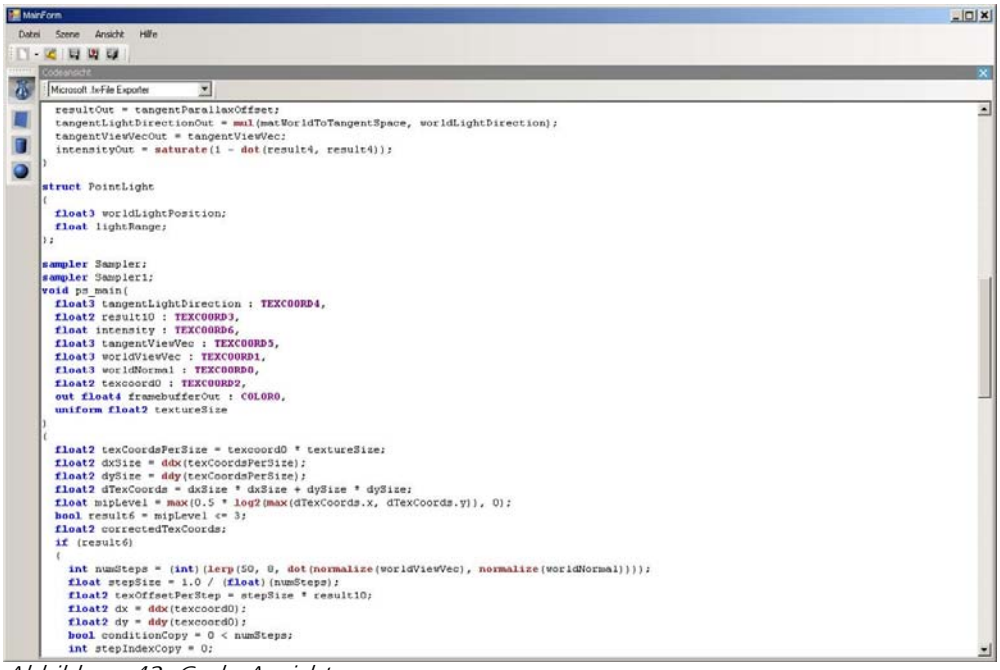


Abbildung 42: Code-Ansicht

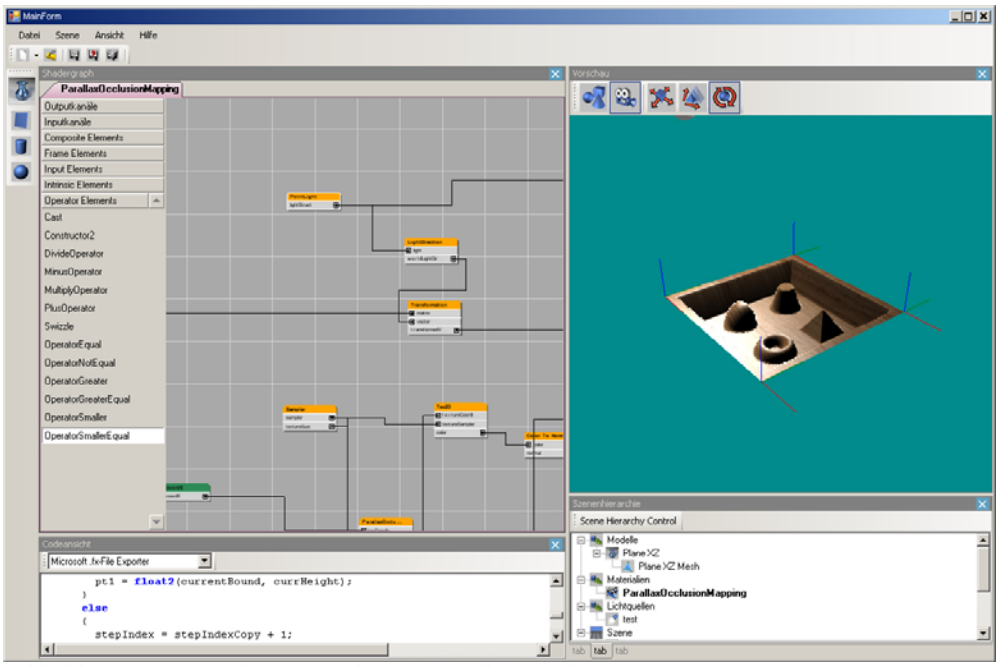


Abbildung 43: Vorschau-Ansicht mit Parallax Occlusion Mapping

3 Interessante Fakten

Es folgen einige interessante Fakten zum Projekt. Die Zahlen innerhalb der Klammern gelten für das gesamte Projekt, während die anderen Zahlen die beiden Assemblies welche nicht zur Studienarbeit gehören ausschliessen.

Anzahl Quellcodedateien:	ca. 381 (449)
Anzahl Klassen:	ca. 450 (525)
Anzahl Zeilen Quellcode:	ca. 55'376 (70'107)
Davon Leer- bzw. Kommentarzeilen:	ca. 9'666 (12'732)
Anzahl Unit Tests:	175 (175)

Kapitel IX: Test

1 Test Benutzeroberfläche

Nachdem am Ende der ersten Iteration eine erste Version der Benutzeroberfläche fertig implementiert war, wurden einige Testpersonen hinzugezogen. Diesen wurde mehrmals die Benutzeroberfläche gezeigt. Zuerst nur mit den beiden ersten Varianten. Später kamen auf Vorschläge der befragten Personen neue Varianten hinzu. Diese wurden Umgesetzt und ebenfalls getestet. Ein Resultat dieses Tests war der Entscheid, das ToolBox-Steuerelement für die Benutzeroberfläche zu verwenden.

2 Test Shadergraph-Plugin

2.1 Testprogramm

Um das Interface zum Shadergraph-Plugin richtig testen zu können, wurde eine Testapplikation geschrieben. Diese Konsolenapplikation ermöglicht es, einen Shadergraphen über Eingaben auf der Konsole zu erstellen und danach in einen Shader umzuwandeln. Der generierte Shader kann auf der Konsole betrachtet oder als Datei abgespeichert werden. Während der Entwicklung des Testprogrammes konnten Designlücken im Interface entdeckt und behoben werden.

2.2 Darstellung

Die während der Entwicklung des Prototypen generierten Shader werden entweder in „ATI Render Monkey“ oder in „nVidia FX Composer“ dargestellt. Diese Programme erlauben es, wie bereits in früheren Kapiteln erwähnt, Shader zu entwickeln. Die Wahl fiel dabei auf „ATI Render Monkey“, da die Erfassung von Szenen, das heisst das Laden von Modellen und den dazugehörenden Texturen, in nVidia FX Composer nicht innerhalb einer Stunde Aufwand funktioniert hat, während „ATI Render Monkey“ nach einer halben Stunde Einarbeitungszeit einsatzbereit war.

Mit Hilfe des erstellten Testprogrammes wurden einige Shader generiert und in eine Datei gespeichert. Diese generierten Shader konnten danach in ATI Render Monkey getestet werden. In Abbildung 44 wird das Resultat eines Shaders gezeigt.

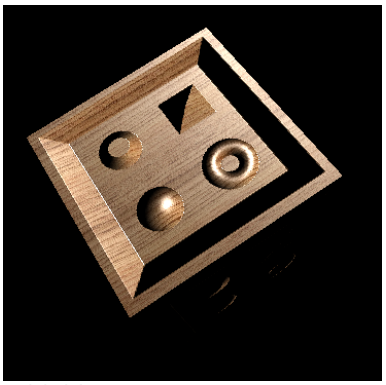


Abbildung 44: Test eines Parallax-Shaders

Durch diese Tests konnte sichergestellt werden, dass der generierte HLSL Code vom DirectX HLSL Compiler auch kompiliert werden kann. Zusätzlich konnten einige Fehler wie falsche Variabeldeklarationen und falsche Vektortransformationen gefunden und behoben werden.

Es existieren Darstellungstests für die folgenden Standardeffekte:

q) Lambert Diffuse
r) Phong Specular
s) Kombiniertes Lambert Diffuse und Phong Specular
t) Normal Mapping
u) Parallax Mapping

2.3 Unit-Tests

Unit-Tests überwachen, dass Klassen wie beabsichtigt funktionieren und sich im Falle von Refactoring keine neuen Fehler einschleichen. Sie sind insbesondere sehr gut dazu geeignet unbeabsichtigte Seiteneffekte zu finden und zu beheben. Die Unit Tests für das Shadergraph-Plugin befinden sich im Assembly

`Shaded.DomainLayer.Material.Shadergraph.Plugin.Hlsl30.Test.exe`. Die Namensräume in welchen sich die Testfälle befinden sind identisch zu den Namensräumen der gesteten Klassen und unter dem Namensraum `Shaded.DomainLayer.Material.Shadergraph.Plugin.Hlsl30.Test` organisiert.

3 Funktionale Anforderungen

In diesem Abschnitt wird geprüft ob die Software die gestellten funktionalen Anforderungen erfüllt. Die folgenden Tabellen geben einen Überblick:

3.1 FA1: Material serialisieren

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA1a: Neues Material erstellen	UC1	1	I.O.
FA1b: Material entfernen	UC5	1	I.O.
FA1c: Material laden	UC2	1	I.O.
FA1d: Material speichern	UC3	1	I.O.
FA1e: Material exportieren	UC4	1	I.O.

3.2 FA2: Modell importieren

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA2a: Modell importieren	UC6	1	I.O.
FA2b: Modell entfernen	UC12	1	I.O.

3.3 FA3: Szene serialisieren

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA3a: Szene laden	UC8	1	Teilweise I.O.
FA3b: Szene speichern	UC9	1	I.O.

3.4 FA4: Szene verändern

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA4a: Modell rotieren	UC11	2	I.O.
FA4b: Modell verschieben	UC11	2	I.O.
FA4c: Ansicht um Szene rotieren	UC13	1	I.O.
FA4d: Ansicht in Szene verschieben	UC13	1	I.O.
FA4e: Ansicht an Szene heran oder davon weg zoomen	UC13	1	I.O.
FA4f: Der Benutzer kann Materialien Modellen zuordnen	UC14	1	I.O.
FA4g: Lichtquelle einfügen	UC15	1	I.O.
FA4h: Lichtquelle verändern	UC16	2	I.O.
FA4i: Lichtqulle entfernen	UC17	2	Nicht I.O.
FA4j: Szene zurücksetzen	UC10	1	I.O.

3.5 FA5: Material/Shadergraph ändern

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA5a: Material zum Editieren wählen	UC18	1	I.O.
FA5b: Element in Shadergraph einfügen	UC19	1	I.O.

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA5c: Eigenschaften der Elemente verändern	UC20	1	I.O.
FA5d: Elemente verknüpfen	UC21	1	I.O.
FA5e: Elemente löschen	UC22	1	I.O.
FA5f: Shadergraph verlassen	UC23	1	I.O.

3.6 FA6: Shadergraph definieren und konvertieren

Teilanforderung	Use-Case	Priorität	Erfüllt?
FA6a: Shadergraph Plugin Interface	-	1	I.O.
FA6b: Shadergraph Plugin	-	1	I.O.

Kapitel X: Schlussfolgerungen

1 Zusammenfassung

Nach den ersten beiden Wochen, in welchen die Dokumentation vorbereitet wurde und der erste Prototyp für die Benutzeroberfläche erstellt wurden ging die Arbeit rasant voran. Durch jeweils kleine Änderungen an Design und Code konnte Schrittweise das Shadergraph-Plugin entwickelt werden. Das Ziel war, soviel Funktionalität zu integrieren, dass die Shader der ersten zehn Kapitel im Buch „Programming Vertex and Pixel Shaders“ von Wolfgang Engel ([WENG01]) mit dem Shadergraphen konstruiert und generiert werden können. Dadurch waren mit jedem Kapitel auch die Erweiterungen des Plugins vorgegeben und leicht umsetzbar. Nach dem guten Start mit einem Zeitgewinn von einer Woche wurde sehr schnell ersichtlich, dass die kleineren Dinge Zeit benötigen würden. Dazu gehörten vor allem die nicht in die Zeitplanung mit einbezogenen Elemente wie die Code-Ansicht oder die Szene-Hierarchie. Trotz der Verzögerungen, wiederum im Rahmen einer Woche, konnte der letzte Prototyp termingerecht fertiggestellt werden.

Abschliessend lässt sich sagen, dass die gesteckten Projektziele erreicht wurden. Der entstandene Prototyp kann sehr gut als „Proof of Concept“ betrachtet und getestet werden. Trotz der noch existierenden Mängel und der vielen fehlenden Details entsteht ein guter Eindruck von der Leistungsfähigkeit des Produktes. Es kann angenommen werden, dass der Schritt vom Prototypen zum fertigen Produkt in ungefähr nochmals 200 bis 300 Stunden machbar sein müsste.

Die gefundenen Verbesserungsmöglichkeiten wurden dokumentiert und könnten in einer Fortsetzungsarbeit umgesetzt werden.

2 Erreichte Ziele

Es wurden praktisch alle funktionalen Anforderungen erreicht. (Siehe auch die Tabelle im Kapitel IX) Die Anforderungen bezüglich Stabilität wurden ebenfalls erreicht. Die Performance-Ziele konnten jedoch nicht ganz erreicht werden. Das Einfügen von Elementen in den Shadergraphen liegt manchmal ein wenig über dem Zeitlimit der unmittelbaren Tätigkeit. Um dieses Problem zu beheben, müsste die Applikation mithilfe eines Profilers analysiert und verbessert werden.

Die folgenden zusätzlichen Features konnten realisiert werden:

- Teilweise Globalisierung: Die Applikation ist zu einem grossen Teil globalisiert. Das heisst sie kann grundsätzlich in verschiedene Sprachen übersetzt werden.
- Anzeigen von lokalen Koordinatensystemen bei Modellen: Es können die lokalen Koordinatensysteme für jeden einzelnen Vertex eines Modells in der Vorschau-Ansicht eingeblendet werden.
- Skalieren eines Objektes
- Frames: Das Konzept der Frames im Shadergraphen war in der ursprünglichen Aufgabenstellung nicht vorgesehen, konnte jedoch realisiert werden.

3 Ausblick

Bei der vorliegenden Version handelt es sich wie von Anfang an geplant um einen Prototypen. Dadurch bestehen viele Erweiterungsmöglichkeiten, die in diesem Kapitel erläutert werden.

Neben den dokumentierten Erweiterungsmöglichkeiten verbleiben auch Bugs. Dazu gehören einige bekannte und sehr wahrscheinlich auch noch unbekannte Bugs. Diese zu beheben sollte nicht schwierig sein. Allerdings ist der Aufwand dafür schwierig abzuschätzen.

3.1 Globalisierung

Es sollte sichergestellt werden, dass die Applikation komplett globalisiert ist. Dazu gehört auch das Globalisieren des Shadergraph-Plugins. Dieses Plugin ist noch nicht komplett globalisiert.

3.2 Verbesserung der Benutzeroberfläche

Die Benutzeroberfläche kann noch weiter verbessert werden. Dazu gehören die folgenden Punkte:

- Durchführung von Usability-Tests oder ähnlichen Tests. Dazu gehört auch, die Software professionellen Grafikern vorzustellen und sie nach ihrer Meinung zu befragen. Darauf aufbauend könnte die Benutzeroberfläche mit Sicherheit verbessert werden.
- Einführen von Tasten-Kürzeln. Mit Tasten-Kürzeln würde die Benutzeroberfläche vor allem für den erfahrenen Benutzer viel einfacher und schneller bedienbar.

3.3 Verbesserung der Szene

In der Vorschau-Ansicht könnten weitere Features hinzugefügt werden. Dazu gehören die folgenden Punkte:

- Hinzufügen von Schatten: Ein Benutzer sieht die dreidimensionale Wirkung eines Bildes besser, wenn es Schatten enthält. Die Szene könnte so modifiziert werden, dass Modelle auf Wunsch des Benutzers einen Schatten werfen.
- Konzept des Renderers: Im aktuellen Design wird die Szene von der Klasse `Scene` und den von `SceneNode` abgeleiteten Klassen gerendert. Um eine bessere Vielfalt und Erweiterbarkeit zu erreichen, sollte das Konzept des Renderers von der Szene entkoppelt werden.
- High Dynamic Range Rendering [WIKI09]: Diese spezielle Technik des Renderings ermöglicht es grosse Unterschiede in der Helligkeit von Beleuchtungen darzustellen. Dadurch sieht das resultierende Bild realistischer aus. Es ist jedoch so, dass es nicht reicht, wenn ein Shader das High Dynamic Range Rendering (kurz HDR Rendering) unterstützt. Daher könnte als Erweiterung ein Renderer implementiert werden, der es ermöglicht, dass der Benutzer damit Shader mit Unterstützung des HDR Rendering entwickeln kann.
- Deferred Shading: Wie das High Dynamic Range Rendering ist das Deferred Shading eine Technik die von Shader und Renderer unterstützt werden muss. Diese Erweiterung der Software wäre ebenfalls denkbar.

3.4 Allgemeine Verbesserungen

Die in diesem Abschnitt gezeigten Erweiterungen betreffen nicht eine einzelne Komponente von shaded, sondern würden Anpassungen in einigen, wenn nicht allen Komponenten:

- Unterstützung von Animationen: Die Unterstützung der Entwicklung von Animationen benötigt Änderungen im Shadergraph und der Szene. Die Erweiterung ist aufwändig, jedoch unterteilbar in die verschiedenen Animationstechniken. Denkbar wären zum Beispiel triviale Animationen durch die Übergabe einer Zeit-Konstante an den Shader oder skeletale Animation von Modellen.

3.5 Umsetzung als Plugin

Eine Alternative zur entwickelten Benutzeroberfläche könnte shaded direkt als Plugin in eines der verbreiteten Modellierungsprogramme integriert werden. Dafür müsste die Benutzeroberfläche ersetzt werden. Das DomainLayer hingegen könnte grösstenteils unverändert übernommen werden.

Teil D - Anhang

Anhang I: Glossar

Shadergraph	Der Begriff Shadergraph wurde in dieser Studienarbeit eingeführt. Mit ihr wird eine Graphenstruktur bezeichnet aus der ein Shader generiert werden kann.
Shader	Mit dem Begriff Shader wird ein Programm bezeichnet, welches beim Rendering abhängig von Eingabewerten die Oberfläche eines Modelles rendert.
Shaderhochsprache	Eine Shaderhochsprache ist eine Programmiersprache in welcher Shader entwickelt werden können. Shaderhochsprachen sind ähnlich zu programmieren wie Beispielsweise C.
HLSL	HLSL steht für „High Level Shading Language“. HLSL wurde von Microsoft entwickelt und mit DirectX 9.0 eingeführt. HLSL ist wie cg und GLSL eine Shaderhochsprache.
Cg	Die Shaderhochsprache Cg war die erste veröffentlichte Shaderhochsprache für Grafikkarten. Sie wurde von nVidia entwickelt.
GLSL	Die „OpenGL Shading Language“ ist die Shaderhochsprache die in OpenGL bevorzugt verwendet wird.
Vertexshader	Der Begriff Vertexshader stammt aus DirectX. Er bezeichnet einen auf Eckpunkten (in Englisch Vertices) operierenden Shader.
Pixelshader	Der Begriff Pixelshader stammt aus DirectX. Er bezeichnet einen auf einzelnen Bildpunkten (Pixeln) operierenden Shader.

Anhang II: Literaturverzeichnis

[ATI01]: ATI Technologies Inc., **RenderMonkey**; http://www.ati.com/developer/rendermonkey/index.ht, Juni 2006

[CLAR01]: Craig Larman, **Applying UML and Patterns**, ISBN: 0-13-148906-2,

[CLAR01]: Craig Larman, **Applying UML and Patterns**, ISBN: 0-13-148906-2,

[COPR01]: Aju George, **Another ToolBox Control**; http://www.codeproject.com/cs/miscctrl/AgToolBox.a, Juni 2006

[GDEV01]: Kelly L. Murdock, **3ds max 7**; http://www.gamedev.net/features/reviews/productrev, April 2006

[GDEV02]: Kelly L. Murdock, **3ds max 8 Review**; http://www.gamedev.net/features/reviews/productrev, April 2006

[GDEV03]: John Hattan, **Cinema 4D R9**; http://www.gamedev.net/features/reviews/productrev, April 2006

[GDEV04]: Kelly L. Murdock, **Maya 6 Review**; http://www.gamedev.net/features/reviews/productrev, April 2006

[GDEV05]: Bryan Mau, **RenderMonkey version 1.5 (build 424)**; http://www.gamedev.net/features/reviews/productrev, April 2006

[NATA01]: Natalya Tatarchuk, **Parallax Occlusion Mapping**; http://www.ati.com/developer/gdc/Tatarchuk-Paralla, Juni 2006

[NVID01]: nVidia Corporation, **FX Composer 1.8**; http://developer.nvidia.com/object/fx_composer_hom, Juni 2006

[NVIDIA01]: nVidia Corporation, **FX Composer**; http://developer.nvidia.com/object/fx_composer_hom, Juni 2006

[RBOR01]: Ralf Borau, **Entwicklung einer XML-basierten visuellen Shading Language**; http://wwwcs.uni-paderborn.de/cs/ag-domik/diplomar, Juni 2006

[WENG01]: Wolfgang Engel, **Programming Vertex and Pixel Shaders**, ISBN: 1-58450-349-2, 2005

[WIKI02]: Wikipedia Community, **3D Studio Max**; http://en.wikipedia.org/wiki/3ds_max, April 2006

[WIKI03]: Wikipedia Community, **Cinema 4D**; http://en.wikipedia.org/wiki/Cinema4d, April 2006

[WIKI04]: Wikipedia Community, **Maya (software)**; , April 2006

[WIKI05]: Wikipedia Community, **RenderMan**; http://en.wikipedia.org/wiki/RenderMan_Interface_S, April 2006

[WIKI06]: Wikipedia Community, **Morphing**; http://en.wikipedia.org/wiki/Morphing, Juni 2006

[WIKI07]: Wikipedia Community, **Skeletal Animation**; http://en.wikipedia.org/wiki/Skeletal_animation, Juni 2006

[WIKI08]: Wikipedia Community, **Displacement Mapping**; http://en.wikipedia.org/wiki/Displacement_Mapping, Juni 2006

[WIKI09]: Wikipedia Community, **High Dynamic Range Rendering**; http://en.wikipedia.org/wiki/High_dynamic_range_re, Juni 2006

[WIKI10]: Wikipedia Community, **Blender**; http://de.wikipedia.org/wiki/Blender_%28Software%2, Juni 2006

Anhang III: Persönlicher Bericht

Als ich vor gut zwei Jahren rtZen/rtshader entdeckte war ich von dessen Fähigkeiten begeistert. Ich konnte das Programm installieren, starten und ohne eine Hilfe zu erstellen Shader erstellen. Später stellte ich fest, dass es trotzdem ein wenig Übung braucht um mit rtshader etas „Sinnvolles" zu erstellen. Vor allem waren die Möglichkeiten relativ beschränkt, da die Elemente des Shadergraphen in rtshader fixe Algorithmen implementierten. Ein Beispiel dafür war die Algorithmen zur Beleuchtung von Objekten. Mit rtshader erstellte Shader besassen immer den gleichen Beleuchtungsalgorithmus. Es war dem Benutzer nicht möglich eigene Beleuchtungsalgorithmen zu implementieren. Ich hatte schon damals das Gefühl, dass ich die Funktionalität besser und für den Benutzer freier bedienbar implementieren könnte. Jedoch fehlte mir neben dem Studium die Zeit meine Ideen zu planen und umzusetzen. Die Gelegenheit dazu kam mit der zweiten Studienarbeit als klar war, dass ich die Arbeit alleine machen musste und ich kurzfristig Urs-Martin Künzi als meinen Betreuer gewinnen konnte.

Die Arbeit entwickelte sich zu Beginn planmässig. Meine Idee das Shadergraphplugin schrittweise nach dem Buch „Programming Vertex and Pixel Shaders" ([WENG01]) erwies sich als sehr gut. Ebenso die Wahl der Programmiersprache. Beides in Kombination erlaubte mir das Plugin schrittweise jeweils um relativ kleine und dadurch einfach umzusetzende Fähigkeiten zu erweitern. Das Buch gab mir dabei eine gute Vorlage der Verbesserungen und mit C# war das dafür nötige Refactoring sehr einfach durchzuführen. Meine Zeitplanung erwies sich dabei ebenfalls als grosszügig bemessen wodurch es mir möglich war eine Woche einzusparen. Später benötigte ich diese Woche für die Vorschau-Ansicht und die in der Planung vergessenen Elemente der Benutzeroberfläche. Vor allem die Dokumentation erwies sich als sehr aufwändig und mühsam da ich alles selbst schreiben musste. Als Vergleich kann meine letzte Studienarbeit dienen. Dort verteilte sich der Aufwand für eine ebenso aufwändige Dokumentation auf drei Personen. Dazu kamen die Klassendiagramme welche zumeist viel zu Gross (das Spektrum reichte bis zu einer Grösse von über einem Quadratmeter) waren und daher nicht einfach so in die Dokumentation eingefügt werden konnten. Ich löste das Problem indem ich die Diagramme stark vereinfachte.

Alles in allem bin ich mit dem Resultat sehr zufrieden. Ich habe einige interessante zusätzliche Features implementiert und dabei mit einer Ausnahme die Anforderungen erfüllt. Natürlich enthält die endgültige Version der Studienarbeit diverse Mängel und Fehler. Der Aufwand um alle diese Probleme zu finden und beheben liegt jedoch Schätzungsweise noch einmal im Rahmen von etwa sechs Wochen. Trotzdem kann das entwickelte Programm sehr gut als „Proof of Concept" dienen und zeigt das dahintersteckende Potential.

Anhang IV: Dank

Ich bedanke mich für die freundliche Unterstützung bei folgenden Personen:

Simon Felix	Für eine ausführliche Diskussion über den User interface Prototypen und neue Ideen zu dessen Gestaltung.
Benjaming Sigg	Für das Testen des User Interface Prototypen und seine Vorschläge zu dessen Verbesserung.
Jonas Schwertfeger	Für das Prüflesen der Dokumentation.
Basil Fierz	Für eine frühe Version des .ASE-Importers. Zusätzlich hat er mir wertvolle Unterstützung mit C# und Managed DirectX gewährt.
Hansruedi & Barbara Oberholzer	Für das Prüflesen der Dokumentation.

Anhang V: Programmierrichtlinien

1 Code Organisation

Generell gilt die Regelung, dass pro Klasse eine .cs-Datei erstellt werden soll.

2 Kommentare

Grundsätzlich soll der Kommentar mit den für C# üblichen Xml-Style-Dokumentationstags geführt werden. Visual Studio.NET fügt diese Kommentarelemente selbständig ein sobald vor dem zu kommentierenden Element die Zeichenfolge „///" eingegeben wird. Zusätzlich soll jede .cs-Datei einen Dateiheader mit folgendem Format erhalten:

```
///<file>[DATEINAME]</file>
///<brief>[KURZBESCHREIBUNG]</brief>
///<datelist>
///<date>[DATUM] [AUTOR], [ÄNDERUNG1]</date>
///<date>[DATUM] [AUTOR], [ÄNDERUNG2]</date>
///</datelist>
///<description>
///[LANGE BESCHREIBUNG]
///</description
```

3 Naming

3.1 Variablen

Namen von Variablen werden mit Kleinbuchstaben begonnen. Jedes weitere „Wort" im Variablennamen, ausser dem ersten, beginnt danach mit einem Grossbuchstaben. Beispiel:

```
int myVar;
```

3.2 Klassen und Strukturen

Die Namen von Klassen und Strukturen beginnen mit einem Grossbuchstaben. Jedes weitere „Wort" im Klassennamen beginnt wiederum mit einem Grossbuchstaben. Beispiel:

```
class MyClass
{
}
```

3.3 Methoden

Methoden sollen wie Klassennamen behandelt werden. Das heisst sie beginnen mit einem Grossbuchstaben und jedes weitere „Wort" im Namen beginnt ebenfalls mit einem Grossbuchstaben. Die Ausnahme sind generierte Methoden, welche nicht umbenennt werden müssen. Beispiel:

```
public void DoSomething()
{
}
```

3.4 Enumerationen

Enumerationen und Elemente von Enumerationen sollen wie Klassennamen behandelt werden. Das heisst sie beginnen mit einem Grossbuchstaben und jedes weitere „Wort“ im Namen beginnt ebenfalls mit einem Grossbuchstaben. Beispiel:

```
enum MyEnumeration
{
  FirstValue,
  SecondValue
}
```

3.5 Konstanten

Konstanten sollen komplett in Grossbuchstaben geschrieben werden. Einzelne „Worte“ sollen durch „_“-Zeichen voneinander getrennt werden. Beispiel:

```
const int MY_CONSTANT = 3;
```

4 Expressions und Statements

Wie Expressions und Statements genau geschrieben werden, kann jeder selber entscheiden.

5 Exceptionhandling

Für die Fehlerbehandlung soll konsequent mit Exceptions gearbeitet werden. Exceptions sollen korrekt abgefangen und behandelt werden.

Anhang VI: Sitzungsprotokolle

1 Sitzungsprotokoll, 10. April 2006

1.1 Sitzungsinformationen

Projekt: shaded

Woche: 2

Datum: Montag, den 10. April 2006

Zeit: 16:15 – 16:45

Traktanden

- Stand der Arbeit
- Beschaffung passender Hardware
- Sitzungstermine

1.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Dokumentation wurde gezeigt. Es wurde beschlossen, dass Formalitäten nur eingehalten werden wo es Sinn macht, und daher Dokumente wie zum Beispiel Use-Cases im „Fully Dressed"-Format, Systemsequenzdiagramme und Contracts nur geschrieben werden, wenn es auch wirklich einen Nutzen bringt.
- Der Betreuer möchte regelmässig, das heisst etwa einmal pro Woche, über den Stand der Arbeiten informiert werden.

Beschaffung passender Hardware

- Das Problem, dass der Rechner am Arbeitsplatz im Studienzimmer eine Grafikkarte enthält welche für die Arbeit nicht ausreicht wurde besprochen. Urs-Martin Künzi wird sich darum kümmern, dass dieses Problem gelöst, und eine ausreichend gute Grafikkarte bereitgestellt wird.

Sitzungstermine

- Es wurde beschlossen, dass Sitzungen ungefähr alle zwei Wochen abgehalten werden. Die Termine werden nicht im Voraus, sondern jeweils für jedes Treffen einzeln per Email festgelegt.

2 Sitzungsprotokoll, 8. Mai 2006

2.1 Sitzungsinformationen

Projekt: shaded

Woche: 6

Datum: Montag, den 8. Mai 2006

Zeit: 16:15 – 16:45

Traktanden

- Stand der Arbeit
- Probleme, Beschaffung Grafikkarte

2.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Arbeit wurde demonstriert und das weitere Vorgehen besprochen.

Probleme, Beschaffung Grafikkarte

- Die für Teile der Studienarbeit benötigte Hardware wurde von der HSR noch immer nicht zur Verfügung gestellt. Aus unbekanntem Grund blieb die Anfrage bei Sven Kosik liegen. Mit nochmaligem Nachfragen sollte das Problem gelöst sein.

3 Sitzungsprotokoll, 22. Mai 2006

3.1 Sitzungsinformationen

Projekt: shaded

Woche: 8

Datum: Montag, den 22. Mai 2006

Zeit: 12:00 – 12:30

Traktanden

- Stand der Arbeit
- Nächste Sitzung

3.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Arbeit wurde demonstriert und das weitere Vorgehen besprochen. Ernsthafte Probleme sind keine vorhanden.

Nächste Sitzung

- Weil in zwei Wochen (5. Juni) Pfingstmontag ist, soll die nächste Sitzung erst am 12. Juni stattfinden.

4 Sitzungsprotokoll, 19. Juni 2006

4.1 Sitzungsinformationen

Projekt: shaded

Woche: 12

Datum: Montag, den 19. Juni 2006

Zeit: 16:00 – 16:30

Traktanden

- Stand der Arbeit
- Nächste Sitzung

4.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Arbeit wurde demonstriert und das weitere Vorgehen besprochen. Ernsthafte Probleme sind keine vorhanden.

Nächste Sitzung

- Die nächste Sitzung ist, da das Ende der Arbeit naht, bereits auf den 26. Juni angesetzt.

5 Sitzungsprotokoll, 26. Juni 2006

5.1 Sitzungsinformationen

Projekt: shaded

Woche: 13

Datum: Montag, den 26. Juni 2006

Zeit: 16:00 – 16:30

Traktanden

- Stand der Arbeit
- Nächste Sitzung

5.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Arbeit wurde demonstriert und das weitere Vorgehen besprochen. Eines der bestehenden Probleme ist, dass der Trackball zwar funktioniert, jedoch vollkommen korrekt. Über einen Trackball werden die Ansicht auf die Szene und die Objekte in der Szene rotiert. Das Problem liegt vermutlich in aufsummierten Ungenauigkeiten bei der Verwendung von Fliesskommazahlen.

Nächste Sitzung

- Die nächste Sitzung ist am 3. Juli 2006 geplant.

6 Sitzungsprotokoll, 5. Juli 2006

6.1 Sitzungsinformationen

Projekt: shaded

Woche: 14

Datum: Mittwoch, den 5. Juli 2006

Zeit: 13:00 – 14:00

Traktanden

- Stand der Arbeit
- Theorie der Quaternionen
- Verschiebung der Abgabe
- Nächste Sitzung

6.2 Diskussion / Beschlüsse

Stand der Arbeit

- Der aktuelle Stand der Arbeit wurde demonstriert und das weitere Vorgehen besprochen. Probleme sind keine mehr aufgetaucht.

Theorie der Quaternionen

- Der Betreuer hat in einer ca. 45 Minütigen Einführung die Theorie der Quaternionen und deren Verwendung für Rotationen im 3-Dimensionalen Raum erklärt.

Verschiebung der Abgabe

- Die Abgabe der Arbeit wurde auf den Morgen des 10.7.2006 aufgeschoben..

Nächste Sitzung

- Diese Sitzung war Letzte im Verlauf der Studienarbeit.

Anhang VII: Zeitplanung

1 Einführung

Es wäre sehr aufwändig Software wie zum Beispiel Microsoft Project an die Gegebenheiten einer Studienarbeit anzupassen. Eines der Probleme sind zum Beispiel die unregelmässigen Arbeitszeiten, da solche Produkte auf Firmen und eine fixe Anzahl Arbeitsstunden pro Tag und Angestelltem ausgelegt sind. Aus diesem Grund wurde bei der vorliegenden Arbeit darauf verzichtet eine solche Software einzusetzen. Stattdessen wurde die Zeitplanung über OpenOffice.org Calc und einige miteinander verlinkte Tabellenblätter geführt. Es folgt zuerst eine Übersicht die geplante und benötigte Zeit aller Phasen. In den darauf folgenden Abschnitten werden die Dateilplanungen der einzelnen Phasen aufgezeigt.

2 Übersicht

	Zeitplanung		Aufwand	Kontrolle
Phasen			SOLL Total Std.	IST Total Std.
Elaboration 1: Analyse & UI Design	03.04.06	16.04.06	41.5	48
Construction 1: Konvertierung	17.04.06	07.05.06	76	62
Construction 2: Graph-Steuerelement	08.05.06	04.06.06	42	43.25
Construction 3: Export	05.06.06	11.06.06	13.5	13.25
Construction 4: Vorschau-Steuerelement	12.06.06	25.06.06	62.25	58.25
Zeitreserve & Dokumentation	26.06.06	07.07.06	36.36	37
Total			271.61	261.75

3 Elaboration Phase 1: Analyse & UI Design

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 03.04.	04.04.	05.04.	06.04.	07.04.	08.04.	09.04.	10.04.	11.04.	12.04.	13.04.	14.04.	15.04.	16.04.
Dokumentation	**22**	**21.5**	**3**	**9**	**0.5**	**3**	**4**	**0**	**0**	**2**	**0**	**0**	**0**	**0**	**0**	**0**
Template für Dokumentation	3	3	2	1												
Projektplan	5	5	1	4												
Anforderungsspezifikation	5	4		4												
Programmierrichtlinien	1	0.5			0.5											
Domainanalyse	5	7				3	4									
Zeitplanung	3	2								2						
UI Prototyp	**19.5**	**26.5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**11**	**7**	**2**	**0.5**	**0**	**0**
Evaluation	6	7								5		2				
Persona	0.5	0														
UI Entwurf	3	6.5								1	4		1	0.5		
Design	3	4									4					
Implementierung	5	7									3	3	1			
Tests	2	2										2				
Risiken	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS1: Konvertierungskomponente	0	0														
RS2: Vorschauansicht	0	0														
RS3: Ausfall Testystem	0	0														
RS4: Datenverlust	0	0														
RS5: Ausfall Teammitglied	0	0														
Total	**41.5**	**48**														

4 Construction Phase 1: Konvertierung

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 14.04.	15.04.	16.04.	17.04.	18.04.	19.04.	20.04.	21.04.	22.04.	23.04.	24.04.	25.04.	26.04.	27.04.	28.04.	29.04.	30.04.
Dokumentation	**4**	**3**	**0**	**0**	**0**	**2**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
Dokumentation	4	3				2							1						
Sitzung mit Betreuer	0	0																	
Konvertierungsplugin	**28**	**35**	**2**	**5**	**10**	**3**	**7**	**4**	**2**	**0**	**0**	**0**	**2**	**0**	**0**	**0**	**0**	**0**	**0**
Design	6	8	2	1	3	1	1												
Implementierung	20	26		4	7	2	6	4	2				1						
Test (Shaderdarstellung)	2	1											1						
Test (Unittests)	0	0																	
Test (Konsolenapplikation)	0	0																	
Risiken	**3.5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS1: Konvertierungskomponente	3.5	0																	
RS2: Vorschauansicht	0	0																	
RS3: Ausfall Testystem	0	0																	
RS4: Datenverlust	0	0																	
RS5: Ausfall Teammitglied	0	0																	

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 01.05.	02.05.	03.05.	04.05.	05.05.	06.05.	07.05.	08.05.	09.05.	10.05.	21.05.	22.05.
Dokumentation	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0.5**	**0**	**0**	**0**	**0.5**
Dokumentation	4	8		3						5				
Sitzung mit Betreuer	1	1								0.5				0.5
Konvertierungsplugin	**36**	**23**	**3**	**0**	**0**	**4**	**3**	**2**	**0**	**0**	**4.5**	**1.5**	**2**	**3**
Design	2	0												
Implementierung	8	5											2	3
Test (Shaderdarstellung)	6	3	3											
Test (Unittests)	10	9				4	3	2						
Test (Konsolenapplikation)	10	6									4.5	1.5		
Risiken	**3.5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS1: Konvertierungskomponente	3.5	0												
RS2: Vorschauansicht	0	0												
RS3: Ausfall Testystem	0	0												
RS4: Datenverlust	0	0												
RS5: Ausfall Teammitglied	0	0												
Total	**76**	**62**												

5 Construction Phase 2: Graph-Steuerelement

	Aufwand	Kontrolle	Tatsächliche Arbeitszeit (in Std. Und pro Tag)													
Arbeitspakete	SOLL Total Std.	IST Total Std.	10.05.	11.05.	12.05.	13.05.	14.05.	15.05.	16.05.	17.05.	18.05.	19.05.	20.05.	23.05.	24.05.	25.05.
Dokumentation	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Dokumentation	0	0														
Sitzung mit Betreuer	0	0														
Änderungen Konvertierungsplugin	**11**	**11.5**	**0**	**0$**		**0**	**0**	**2**	**7**	**0**	**0**	**1.5**	**1**	**0**	**0**	**0**
Design	3	2						2								
Implementierung	8	9.5							7			1.5	1			
Prototyp Shadergraph UI	**31**	**31.75**	**4**	**3.5**	**2**	**3**	**1**	**0**	**2**	**2**	**0**	**3**	**3**	**1**	**4.25**	**3**
Design	6	4.5	2.5	0.5					0.5						1	
Implementierung	25	27.25	1.5	3	2	3	1		1.5	2		3	3	1	3.25	3
Risiken	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS1: Konvertierungskomponente	0	0														
RS2: Vorschauansicht	0	0														
RS3: Ausfall Testystem	0	0														
RS4: Datenverlust	0	0														
RS5: Ausfall Teammitglied	0	0														
Total	**42**	**43.25**														

6 Construction Phase 3: Export

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 26.05.	27.05.	28.05.	29.05.	30.05.	31.05.	01.06.	02.06.	03.06.	04.06.
Dokumentation	0	0	0	0	0	0	0	0	0	0	0	0
Dokumentation	0	0										
Sitzung mit Betreuer	0	0										
Änderungen Konvertierungsplugin	4	6	0	0	0	6	0	0	0	0	0	0
Design	0	0										
Implementierung	4	6				6						
Codeansicht	6.5	6	5	0	0	0	1	0	0	0	0	0
Design	0.5	0.5	0.5									
Implementierung	6	5.5	4.5				1					
Export Plugin	2.5	1.25	0	0	0	0	0	0	1.25	0	0	0
Design	0.5	0.25							0.25			
Implementierung	2	1							1			
Eigenschaftenansicht	4.5	6	0	0	0	0	0	0	6	0	0	0
Design	0.5	1							1			
Implementierung	4	5							5			
Risiken	0	0	0	0	0	0	0	0	0	0	0	0
RS1: Konvertierungskomponente	0	0										
RS2: Vorschauansicht	0	0										
RS3: Ausfall Testystem	0	0										
RS4: Datenverlust	0	0										
RS5: Ausfall Teammitglied	0	0										
Total	**13.5**	**13.25**										

7 Construction Phase 4: Vorschau-Steuerelement

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 05.06.	06.06.	07.06.	08.06.	09.06.	10.06.	11.06.	12.06.	13.06.	14.06.	15.06.	16.06.	17.06.	18.06.
Dokumentation	**2**	**1.25**	**0**	**0**	**1**	**0**	**0**	**0**	**0.25**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Dokumentation	2	1.25			1				0.25							
Sitzung mit Betreuer	0	0														
Refactoring	**7**	**11**	**6**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
Redesign	3	3	3													
Refactoring	4	8	3	4												1
Vorschauansicht	**26**	**26.5**	**5**	**2**	**1**	**0**	**4**	**0**	**0**	**0**	**9**	**0**	**3**	**2.5**	**0**	**0**
Design	7	5.5	2				1				2		0.5			
Implementierung Rendering	4	4	2								2					
Implementierung Modelle und Objekte	6	6.5		2							2.5		2			
Implementierung Materialien	4	3.5					1				2.5					
Implementierung Scenegraph	5	4	1		1		2									
Implementierung Lichtquellen	0	3											0.5	2.5		
Implementierung Serialisierung Szene	0	0														
Szenenhierarchie	**8**	**7.5**	**0**	**2.5**	**3**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
Design	2	1		1												
Implementierung	6	6.5		1.5	3		1				1					
Risiken	**1.75**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS 1: Konvertierungskomponente	0	0														
RS 2: Vorschauansicht	1.75	0														
RS 3: Ausfall Testsystem	0	0														
RS 4: Datenverlust	0	0														
RS 5: Ausfall Teammitglied	0	0														

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 19.06.	20.06.	21.06.	22.06.	23.06.	24.06.	25.06.
Dokumentation	**0.5**	**0.5**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Dokumentation	0	0							
Sitzung mit Betreuer	0.5	0.5	0.5						
Refactoring	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Redesign	1	0							
Refactoring	3	0							
Vorschauansicht	**9**	**9.5**	**0.5**	**0**	**1**	**2**	**0**	**0**	**0**
Design	1	0.5	0.5						
Implementation Rendering	0	1			1				
Implementation Modelle und Objekte	0	2				2			
Implementation Materialien	0	0							
Implementation Scenegraph	0	0							
Implementation Lichtquellen	4	0							
Implementation Serialisierung Szene	4	6	1	5					
Szenenhierarchie	**4**	**2**	**0**	**0**	**0**	**0**	**2**	**0**	**0**
Design	0	0							
Implementierung	4	2					2		
Risiken	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS 1: Konvertierungskomponente	0	0							
RS 2: Vorschauansicht	0	0							
RS 3: Ausfall Testsystem	0	0							
RS 4: Datenverlust	0	0							
RS 5: Ausfall Teammitglied	0	0							
Total	**62.25**	**58.25**							

8 Zeitreserve & Dokumentation

Arbeitspakete	Aufwand SOLL Total Std.	Kontrolle IST Total Std.	Tatsächliche Arbeitszeit (in Std. Und pro Tag) 26.06.	27.06.	28.06.	29.06.	30.06.	01.07.	02.07.	03.07.	04.07.	05.07.	06.07.	07.07.	08.07.	09.07.
Dokumentation	**16**	**37**	**5**	**4**	**4.5**	**1**	**4**	**0**	**0**	**0**	**6**	**1**	**1**	**4**	**1**	**4**
Dokumentation	15	35.5	5	4	4.5	1	4				6	1	1	4	1	4
Sitzung mit Betreuer	1	1.5	0.5									1				
Risiken	**20.36**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RS1: Konvertierungskomponente	10.5	0														
RS2: Vorschauansicht	1.75	0														
RS3: Ausfall Testystem	0.3	0														
RS4: Datenverlust	0.01	0														
RS5: Ausfall Teammitglied	7.8	0														
Total	**36.36**	**37**														