# Entwicklung einer GUI für einen Shader Viewer

Bachelorarbeit

von

Christoph Leineweber

geboren am 31. März 1982 in Kronberg i.T.

23. März 2009

Betreuer:

Prof. Dr.-Ing. Detlef Krömker

Dipl. Informatiker Sebastian Schäfer

Goethe Universität Frankfurt am Main

Graphische Datenverarbeitung

Fachbereich Informatik und Mathematik

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Frankfurt am Main, 23. März 2009

---

(Christoph Leineweber)

**Zusammenfassung**

Jede erfolgreiche Software muss in einer geeigneten Art und Weise mit der Person, die sie benutzt, in Verbindung treten. Diese Schnittstelle zwischen Mensch und Maschine ist ein zentraler Baustein in der Softwareentwicklung. Eine noch so mächtige und ausgereifte Software kann ihr Potential nicht ausschöpfen, wenn Probleme und Missverständnisse bei der Kommunikation mit dem Anwender auftreten.
Bei graphischen Benutzeroberflächen erfolgt die Interaktion zwischen Benutzer und technischem System mittels graphischer Symbole, die am Bildschirm dargestellt werden. Die Oberfläche setzt sich aus verschiedenen Menüs und Steuerelementen mit dem Ziel zusammen, die zugrunde liegende Software für den Anwender bedienbar zu machen. Als Eingabegeräte dienen vor allem Maus und Tastatur. Für die Human Computer Interaction oder abgekürzt HCI (Mensch-Computer Interaktion) sind spezielle Normierungen und Anforderungen erstellt worden, die den Entwicklungsprozess unterstützten.
In dieser Arbeit wird eine graphische Benutzeroberfläche für einen Shader Viewer entworfen und implementiert. Beginnend bei ersten Skizzen und Prototypen wird der Entwicklungsprozess bis zur fertigen graphischen Oberfläche dargestellt. Probleme bei der Erstellung werden aufgezeigt und Lösungsstrategien entwickelt. Vor allem spielen Design und Usablity eine entscheidende Rolle. Verschiedene Aspekte und Alternativen, die im Entwicklungsprozess zu beachten sind, werden näher beleuchtet.

**Abstract**

Every successful software has to communicate with its user in some satisfactory kind of way. This interface between humans and machines is a central element in software development. Even powerful and technically mature software cannot exploit its full potential if problems or misunderstandings exist when communicating with the user.
Graphical user interfaces (GUI) interact with the user by displaying graphical symbols on the monitor screen. The GUI is composed of various menus and widgets with the objective of making the respective software usable. The mouse and keyboard are mainly used as input devices. Human-computer interaction (HCI) offers special standardizations and requirements to support the development process.
This thesis describes the concept development and implementation of a GUI for a shader viewer. The development process is described in its entirety - from the initial starting phase with sketches and prototypes to the final version of the GUI. Implementation problems are identified and solution strategies are developed. Design and usability are important aspects that have to be examined. Different facets and alternative solutions in the development process are considered and illustrated.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

In der Informatik spielen Schnittstellen zwischen Mensch und Maschine eine entscheidende Rolle. Um mit interaktiven technischen Systemen arbeiten zu können, ist es immer wieder nötig, Daten an diese zu übermitteln und wieder zu empfangen. Besonders wenn es sich um eine größere Menge komplexer Daten handelt, ist eine geeignete Schnittstelle wichtig und hilfreich. Meist werden graphische Oberflächen entwickelt, die den Bedürfnissen von Benutzern gerecht werden und einen intuitiven Umgang mit dem System erlauben.

## 1.1 Motivation

Wie wichtig und umfangreich die Entwicklung einer geeigneten graphischen Oberfläche ist, wird bereits daraus ersichtlich, dass sich in der Informatik ein eigener Bereich mit dieser Problematik beschäftigt. Dieser Bereich wird als *Human Computer Interaction* (HCI) bezeichnet. Trotz zahlreicher Richtlinien und Leitfäden müssen vom Entwickler Konzepte zur Umsetzung erstellt werden, die an die jeweilige Situation und das spätere Einsatzgebiet der Software angepasst sind.
In dieser Ausarbeitung soll eine graphische Benutzeroberfläche für einen *Shader Viewer*[1] erstellt werden. Einzelne Schritte und Entscheidungen, die während des Entwicklungsprozesses zu beachten sind, werden dazu näher betrachtet und analysiert.
Zu Beginn müssen Überlegungen darüber angestellt werden, welche Voraussetzungen gelten und welche HCI Grundlagen anzuwenden sind. Bevor mit der Programmierung begonnen werden kann, ist es hilfreich, Skizzen und Konzepte zu erarbeiten. Nach Ausarbeitung der Umsetzungsstrategie, kann auf den erstellten Skizzen aufbauend mit der Programmierung begonnen werden.

## 1.2 Überblick über die Aufgabenstellung

Im Fachbereich „Graphische Datenverarbeitung“ wurde Ende letzten Jahres im Rahmen einer Diplomarbeit ein Shader Viewer von Daniel Schiffner entwickelt. Zur Steuerung des Shader Viewers können mit diesem Programm derzeit Benutzereingaben in einer graphischen Oberfläche vorgenommen werden, die sich in einem separaten Fenster befindet. Somit besteht der Viewer aus einem Ausgabefenster und einer vorläufigen graphischen Oberfläche zur Erfassung von Eingabeparametern. Um diesen Viewer später durch eine neue graphischen Oberfläche zu erweitern, wurden von Daniel Schiffner bereits passende Schnittstellen angelegt. Der Zugriff auf die von der Benutzungsschnittstelle benötigten Daten wird dadurch erleichtert.
Die neu entwickelte Benutzungsschnittstelle soll das Arbeiten mit dem Shader Viewer vereinfachen. Das komplette Programm wird später nur aus einem Fenster bestehen, in welches

[1] Programm zum Betrachten von Shader Effekten

die graphische Benutzeroberfläche direkt integriert ist. Eine möglichst intuitive und einfache Bedienung des Shader Viewers soll ermöglicht werden.
Da sehr viele unterschiedliche Eingaben vom Benutzer vorgenommen werden können, wird großer Wert auf Übersichtlichkeit und Strukturierung der Bedienung gelegt. Die Daten müssen geeignet gruppiert und in mehrere Untermenüs gegliedert werden. Da unterschiedlichste Eingabetypen vom Programm verarbeitet werden, muss die graphische Oberfläche für jeden dieser Typen ein geeignetes Eingabeformat bereitstellen. Durch geeignete Unterstützung des Benutzers sollen schon im Vorhinein Fehler bei der Eingabe vermieden werden. Beispielsweise müssen Koordinaten, Winkel, Texturen und Farben korrekt erfasst werden.
Ein weiterer Schwerpunkt der Entwicklung liegt auf einer ästhetischen Optik und einem ansprechenden Design. Dies soll allerdings nicht auf Kosten der Übersichtlichkeit und Usability (Benutzerfreundlichkeit) realisiert werden. Die Schwierigkeit besteht darin, den *Trade Off*[2] zwischen einem ansprechendem Design und einer leichten Bedienbarkeit der späteren graphischen Oberfläche möglichst gut zu lösen.
Ein weiteres Problem beim Erstellen von graphischen Benutzeroberflächen ist, dass Benutzer in unterschiedlichen Betriebssystemen heimisch sind und unterschiedliche Gewohnheiten mitbringen. Entspricht die graphische Oberfläche nicht den Gewohnheiten, kann dies zu Frustration beim Anwender führen. Deshalb wird in dieser Ausarbeitung versucht, einen Kompromiss zwischen den unterschiedlichen Systemen zu finden, auf denen die Anwender gewohnt sind zu arbeiten. Um dies zu erreichen, sollten möglichst intuitive Benutzungsschnittstellen geschaffen werden. Auch wenn eine gesuchte Funktion nicht an der Stelle in der graphischen Oberfläche platziert ist, die ein Anwender gewohnt ist, muss er sich dennoch zurechtfinden und mit geringem Aufwand die gewünschte Funktion finden und ausführen können.
Während der Entwicklung soll die Funktionalität der graphischen Oberfläche durch Benutzerevaluationen auf ihre Korrektheit und Bedienungsfreundlichkeit hin geprüft werden, um eventuellen Mängeln frühzeitig entgegenwirken zu können.

[2] Zielkonflikt, Verbesserung eines Aspekts nur unter Inkaufnahme der Verschlechterung eines anderen.

# 2 Grundlagen

Dieses Kapitel beschäftigt sich mit der Theorie, die zur Erstellung einer graphischen Oberfläche für den Shader Viewer benötigt wird. Es werden Aspekte aus der Computergraphik angesprochen, um ein grundlegendes Verständnis über die Funktionsweise des Shader Viewers zu schaffen. Zusätzlich müssen einige Konzepte der HCI erläutert werden, um spätere Entscheidungen nachvollziehen zu können.

## 2.1 Der Shader Viewer

Die Entwicklung einer graphischen Oberfläche kann nur erfolgreich durchgeführt werden, wenn fundierte Kenntnisse über das zugrunde liegende Programm vorliegen. Dies betrifft nicht nur Informationen über die Schnittstellen sondern auch die Ziele des Programms und die Anwendungsprofile der späteren Nutzer. Daher wird in diesem Abschnitt kurz beschrieben, was ein Shader Viewer ist und wofür ein solches Programm eingesetzt werden kann.

### 2.1.1 Renderpipeline

Als *Rendern* wird in der Computergraphik ein Vorgang bezeichnet, der aus einer Szene ein Bild erzeugt. Eine Szene ist ein im Computer generiertes Modell, das sich aus mehreren Objekten zusammensetzen kann. Diese Objekte haben genaue Positionen, können unterschiedliche Materialeigenschaften aufweisen und werden von einer Lichtquelle beleuchtet. Die komplette Szene wird von einer festgelegten Position aus von einem Beobachter betrachtet. Beim Rendern wird aus all diesen Daten ein zweidimensionales Bild berechnet.
Während des Renderns müssen verschiedene Berechnungen durchgeführt werden, ob beispielsweise die Objekte sich gegenseitig verdecken oder ob das Licht, dass in die Szene fällt, reflektiert wird. Die Berechnung des genauen Aussehens der Oberfläche von Objekten wird als *Shading* bezeichnet.
Weil der Shader Viewer das aktive Modifizieren von Parametern erlaubt und die Betrachtung der Szene aus verschiedenen Richtungen zulässt, kommt bei der Berechnung der Ausgabe das *Echtzeitrendern* zum Einsatz. Dabei müssen die erzeugten Bilder so schnell berechnet werden, dass die Bildfolge vom Benutzer als dynamischer Prozess empfunden wird und eine flüssige Bewegungen entsteht. Der Benutzer sollte nicht auf das Ergebnis seiner Eingaben warten müssen.
Beim Echtzeitrendern werden die ausgegebenen Bilder in der *Renderpipeline* erzeugt. Welche Stationen hierbei durchlaufen werden, ist in Abbildung 1 zu sehen. Diese Darstellung des Renderprozesses ist nur modellhaft und kann in unterschiedlichen Systemen variieren. Sie liefert allerdings einen guten Überblick über die grundlegenden Abläufe beim Rendern. Im Folgenden werden die einzelnen Stationen der Pipeline kurz erläutert.

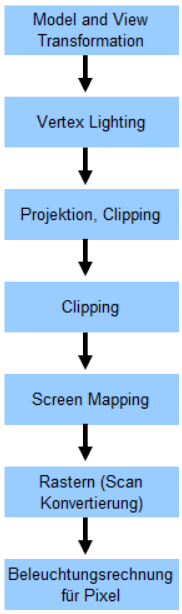


Abbildung 1: Render Pipeline [Krö08]

**Modell- und View-Transformation:** Die Szene, die gerendert werden soll, ist in unterschiedliche Koordinatensysteme aufgeteilt. Objekte befinden sich in ihren eigenen *Objektkoordinaten*, der Beobachter der Szene in *Kamerakoordinaten*. Im ersten Schritt der Renderpipeline werden die Objektkoordinaten in *Weltkoordinaten*, das heißt in die globalen Koordinaten der gesamten Szene, transformiert. Des Weiteren werden die Kamerakoordinaten so transformiert, dass die Kamera sich im Ursprung des Weltkoordinatensystems mit Blickrichtung entlang der Z-Achse befindet.

**Vertex Lighting:** Berechnung der Farbe für jeden Vertex (Knoten). Neben Textur und Materialeigenschaften müssen alle Lichtquellen der Szene berücksichtigt werden. Es muss geprüft werden, welchen Einfluss jede Lichtquelle auf die Farbgebung des jeweiligen Vertex hat.

**Projection:** Das Sichtvolumen, dies ist der sichtbare Bereich, wird in einen Würfel mit der Kantenlänge zwei transformiert. Beim Sichtvolumen kann es sich um ein Frustum (perspektivische Projektion) oder einen Quader (orthogonale Projektion) handeln. Die Tiefeninformation muss für die spätere Verdeckungsrechnung im *Z-Buffer*[3] erhalten bleiben.

**Clipping:** Alle Objekte, die sich außerhalb des Sichtvolumens befinden, werden ausgeschnitten. Sie werden in der weiteren Berechnung nicht mehr berücksichtigt, da sie für den Beobachter nicht sichtbar sind.

**Screen Mapping:** Abbildung der Koordinaten auf dem Bildschirm. Dies geschieht mittels der *Window-Viewport-Transformation*.

**Rastern:** In diesem Schritt wird die Szene gerastert. Alle Pixel werden im *Scan Line*[4] Verfahren nacheinander bearbeitet und eingefärbt. Um die Farbe eines Pixels auf dem Bildschirm zu erhalten, muss vorher eine Verdeckungsrechnung durchgeführt werden. Damit wird ermittelt, welches Objekt für den Beobachter sichtbar ist und somit maßgeblich für die Farbgebung des Pixels ist.

**Beleuchtungsrechnung für Pixel:** In diesem Schritt findet das Shading statt. Je nach angewandtem Verfahren wird berechnet, wie sich die Interaktion von Licht und Material auf die Darstellung des einzelnen Pixel auswirkt. Ein Beispiel hierfür ist das *Phong Shading*[5] zum Erzeugen von *Highlights*. Dies sind Lichtreflexionen oder Glanzlichter, die auf spiegelnden Oberflächen auftreten.

Als Resultat der Renderpipeline liegt das fertig gerenderte Bild vor. [Wat99]

[3] Tiefenpuffer, Verfahren zur Verdeckungsrechnung
[4] Scanline-Algorithmen sind Verfahren in der Computergraphik, die in einem Bild Zeile für Zeile abarbeiten.
[5] Beleuchtungsmodell in der 3D-Computergraphik

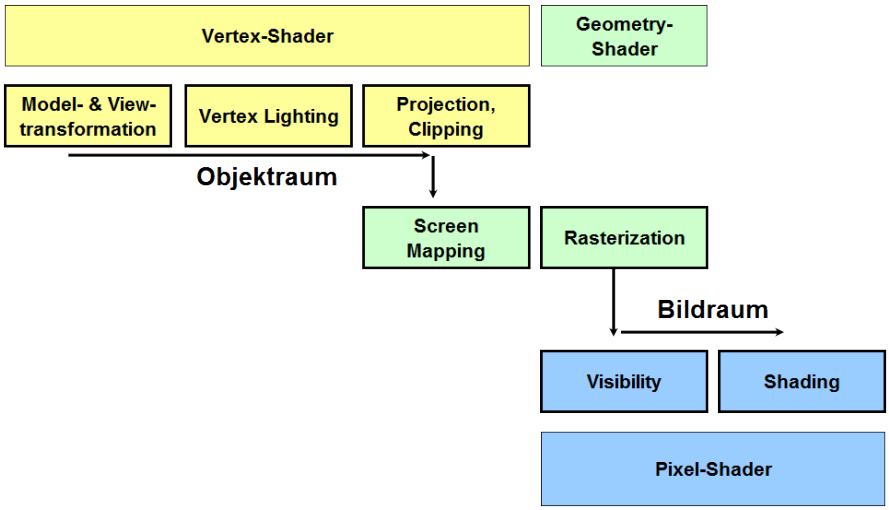


Abbildung 2: Shaderkomponenten in der Render Pipeline [Krö08]

### 2.1.2 Shader

Nachdem im letzten Abschnitt kurz erläutert wurde, wie ein Bild beim Echtzeitrendern entsteht, wird in diesem Abschnitt näher auf das Shading eingegangen.
Als Shader bezeichnet man kleine Hardware- und Softwaremodule, die bestimmte Rendereffekte implementieren. In der Hardware eines Computers werden sie als kleine Recheneinheiten von Graphikchips umgesetzt. Als Software sind Shader kleine Programme, die Teile der Renderpipeline ersetzen. Anfangs wurden Shader nur für das *Pixelshading* verwendet, wie im vorherigen Abschnitt beschrieben. Mit der Weiterentwicklung von *Graphics Processing Units* (Graphikprozessoren) wurden sie auch auf weitere Teile der Pipeline anwendbar. Sie lassen sich in drei verschiedene Kategorien unterteilen: *Vertexshader*, *Pixelshader* und *Geometryshader*. In Abbildung 2 ist zu sehen, in welchen Abschnitten der Renderpipeline die jeweiligen Shadertypen zum Einsatz kommen.
**Vertexshader:** Diese Shader werden für jeden Vertex aufgerufen. Sie berechnen dabei aus den virtuellen dreidimensionalen Positionen der Vertices ihre genaue Position auf der zweidimensionalen Ausgabe und ihren jeweiligen Tiefenwert für den Z-Buffer. Dabei können Position, Farbe oder Texturkoordinaten der Vertices verändert werden. Die berechneten Daten des Vertexshaders werden an den Geometryshader weitergegeben, falls ein solcher vorhanden ist. Ansonsten werden die erstellten Daten vom Pixelshader weiterverarbeitet.
**Pixelshader:** Hier werden die Farben der einzelnen Pixel berechnet und die Objekte der zu rendernden Szene ausgefüllt. Typischerweise werden diese Shader für Beleuchtungseffekte und ähnliche Effekte genutzt. Dazu zählt beispielsweise das *Bump Mapping*, das die Darstellung von detailreichen Oberflächen ohne Veränderung an der Geometrie erlaubt. Dies geschieht

durch Texturen, die durch Modifikation der Normalenvektoren Schattierungen auf der Oberfläche erzeugen. Der Eindruck von dreidimensionalen Strukturen entsteht.
**Geometryshader:** Diese Shader sind relativ neue Elemente der Computergraphik. Wie in Abbildung 2 zu sehen ist, befinden sie sich zwischen Vertex- und Pixelshader. Im Gegensatz zum Vertexshader ist der Geometryshader in der Lage, den zu rendernden Objekten Vertices hinzuzufügen oder von diesen zu entfernen. Dies vermittelt ihm einen Sonderstatus, da das Hinzufügen von Knoten streng genommen nicht Teil des *Rendering Prozesses* ist. Ein Effekt, der mit dieser Art von Shader umgesetzt werden kann, ist beispielsweise das Generieren von *Shadow Volumes*. Dies ist eine spezielle Technik zur Darstellung von Schatten. [FK03] [Ros06]

### 2.1.3 Anwendungsgebiete

Shader Viewer sind Programme zur Visualisierung von Shader Effekten. Solche Programme werden beispielsweise bei der Entwicklung von Computerspielen eingesetzt, um Effekte vor dem Einsatz im Spiel zu testen. Shadereffekte lassen sich auf unterschiedlichen Oberflächen darstellen. Mit der Wahl geeigneter Texturen und Objektstrukturen können Effekte unter ähnlichen Bedingungen simuliert werden, wie sie später im Spiel auftreten.
Shader Viewer können überall dort eingesetzt werden, wo Graphik mit Shadereffekten am Computer erzeugt und dargestellt werden soll.

## 2.2 HCI

*Human Computer Interaction* ist eine Disziplin der Informatik, die sich mit dem Design, der Evaluierung und Implementierung von interaktiven Computer Systemen für die Anwendung durch den Menschen und allen weiteren Wechselbeziehungen, die sich daraus ergeben, beschäftigt [HBC+92]. Die Entwicklung möglichst geeigneter und funktionaler Schnittstellen zur Kommunikation zwischen Mensch und Maschine steht dabei im Vordergrund. In diesem Zusammenhang wurden Richtlinien und Normen entwickelt, die eine der wichtigsten Grundlagen und Anhaltspunkte dieser Ausarbeitung darstellen.

### 2.2.1 Einfluss des Benutzers

Bei der Entwicklung graphischer Oberflächen muss man sich vor allem in das Profil der Benutzer hineinversetzen, die später mit der Software arbeiten sollen. Anhand solcher Betrachtungen kann ein „Durchschnittsbenutzer“ ermittelt und die graphische Oberfläche bestmöglich auf ihn zugeschnitten werden. Oft wird ein Angestellter in einer typischen Bürosituation vorausgesetzt. Kann das Umfeld besser differenziert und können genauere Annahmen getroffen werden, ist es möglich, eine besser angepasste graphische Oberfläche zu entwickeln. [BCW01]
Anwender sind individuell unterschiedlich, jedoch kann anhand von ausgewählten Eigenschaf-

ten versucht werden, eine allgemeingültige Kategorisierung zu erreichen. Hierbei können u.a. Alter, kognitive Fähigkeiten, Geschlecht oder persönliche Vorlieben betrachtet werden. Allerdings ist es schwierig diese Daten zu erfassen, zu klassifizieren und zu bewerten. Zumeist kommen pysiologische, psychologische und soziokulturelle Kategorisierungen zur Anwendung. Zu den physiologischen Aspekten gehört beispielsweise die Fähigkeit des Sehens oder Hörens. Auf der psychologischen Seite sind beispielsweise Intelligenz und Persönlichkeit zu erfassen. Zu den soziokulturellen Faktoren sind Eigenschaften wie Sprache und Kultur zu zählen. [BCW01]
In einigen Fällen kann es sehr einfach sein, die entsprechenden Benutzerinformationen in den Entwicklungsprozess der graphischen Oberfläche einfließen zu lassen. Ein Beispiel hierfür ist die Sprache, in der die graphische Oberfläche für die entsprechenden Anwender implementiert wird. Für andere Aspekte ist es wiederum nicht so einfach, wenn nicht sogar falsch, konkrete Umsetzungshinweise abzuleiten.
Somit kann zusammenfassend festgestellt werden, dass in der HCI Themengebiete aus unterschiedlichen Wissenschaften beachtet werden müssen. HCI ist eine interdisziplinäre Wissenschaft. Um Lösungsstrategien von Benutzern analysieren zu können, sind beispielsweise Kenntnisse aus den Gebieten Psychologie, Medizin und Biologie von Vorteil. Zusätzlich sind Fähigkeiten auf dem Gebiet Graphikdesign nötig, um eine ansprechende graphische Oberfläche zu erstellen. [DFAB98]

### 2.2.2 Benutzerfreundlichkeit

Wie kann man die Benutzerfreundlichkeit (Usability) einer graphischen Oberfläche messen und welche Regeln helfen bei der Erstellung von benutzerfreundlichen Oberflächen?
Um diese Frage zu beantworten, werden im folgen drei Kategorien vorgestellt, die jeweils Prinzipien enthalten, die von der graphischen Oberfläche erfüllt werden sollten.

**I) Erlernbarkeit:** Wie lange brauchen typische Repräsentanten einer Benutzergruppe, um die relevanten Ausführungen der Aufgaben zu erlernen?
**Vorhersagbarkeit:** Der Benutzer sollte immer anhand von vorherigen Eingaben wissen, an welcher Stelle der Software er sich befindet und welche Optionen er hat, sich von hier aus innerhalb des Programms weiter zu bewegen. Er darf nicht von Aktionen des Systems „überrascht“ werden.
**Nachvollziehbarkeit:** Durch geeignete Rückmeldungen der graphischen Oberfläche auf Aktionen des Benutzers kann dieser das Verhalten des Systems besser nachvollziehen.
**Vertrautheit:** Durch Erfahrungswerte aus anderen Programmen oder aus der realen Welt, soll der Benutzer möglichst intuitiv mit der graphischen Oberfläche arbeiten können. Wird ein Button (Knopf) am Bildschirm dargestellt, weiß der Benutzer sofort, dass dieser gedrückt werden kann, da Knöpfe in der Realität genau gleich bedient werden.
**Verallgemeinerung:** Bisher erlernte Lösungsstrategien sollten auch in ähnlich gelagerten Si-

tuationen anwendbar sein.

**Konsistenz:** Beibehaltung von einheitliche Eigenschaften. Dies ist eines der wichtigsten Prinzipien und spiegelt sich in vielen Bereichen wie Farbe, Form und Verhalten wieder.

**II) Flexibilität:** Möglichst weitgehende Berücksichtigung der Präferenzen des Anwenders beim Umgang mit dem System.

**Dialoge öffnen:** Werden viele Dialoge vom System aufgerufen, auf die der Benutzer reagieren muss, bevor er weitere Aktionen ausführen kann, geht Flexibilität verloren. Darauf sollte nur in den nötigsten Fällen zurückgegriffen werden. Vorteilhafter ist es, Dialoge vom Anwender anstoßen zu lassen.

**Multi-Threading:** Möglichkeit mehrere Benutzerinteraktionen gleichzeitig auszuführen.

**Aufgaben übergeben:** In bestimmten Fällen sollte es möglich sein, bei der Abarbeitung einer Aufgabe, dem Anwender zu überlassen, ob er sie selbst ausführen will oder sie vom System bearbeiten lassen möchte. Die Kontrollübergabe während einer Abarbeitung sollte ermöglicht werden.

**Austauschbarkeit:** Unterstützung alternativer Parameter mit gleichen Werten. Dies kann Einheiten (100cm = 1m) oder auch Zahlen betreffen (10+1 = 11).

**Individualisierbarkeit:** Einstellbarkeit der graphischen Oberfläche durch den Benutzer.

**III) Robustheit:** Unterstützung beim Erreichen der Ziele.

**Sichtbarkeit:** Möglichkeit des Anwenders, anhand der von der graphischen Oberfläche dargestellten Information die für ihn aktuell wichtigen Informationen zu entnehmen. Nicht immer kann alles Benötigte dargestellt werden, da beispielsweise der Bildschirm zu klein ist oder die Übersichtlichkeit gewahrt werden soll. Deshalb kann es immer wieder dazu kommen, dass die für den Benutzer gerade wichtigen Informationen nicht direkt sichtbar sind.

**Wiederherstellbarkeit:** Möglichkeit, nach einem Fehler den alten Zustand vor dem Fehler wiederherzustellen. Hierbei wird zwischen *Forward Error Recovery*, bei dem alle Aktionen bis zum Fehler wieder neu ausgeführt werden müssen, und *Backward Error Recovery* unterschieden, bei dem lediglich die letzte Aktion rückgängig gemacht werden muss.

**Antwortzeit:** Wie lange braucht das System, bis es auf eine Aktion des Benutzers reagiert? Diese Zeit sollte kurz und konstant gehalten werden. Treten längere Antwortzeiten auf, sollte das System dem Anwender erkenntlich machen, dass es den Befehl erhalten hat und an dessen Ausführung arbeitet.

**Funktionalität:** Zu welchem Grad werden die vom Benutzer gestellten Anforderungen vom System erfüllt? Gibt es Aufgaben, die vom System nicht bearbeitet werden oder nur teilweise erfüllt werden können? Man unterschiedet zwischen *Task Completeness*, welche angibt, inwieweit die gestellten Aufgaben erfüllt werden und *Task Adequacy*, welche Auskunft darüber gibt, ob die Art und Weise der Bearbeitung mit den Vorstellungen des Anwenders übereinstimmt. [DFAB98]

### 2.2.3 Entwicklungsstrategie LUCID

Um den Entwicklungsprozess einer graphischen Oberfläche zu erleichtern, sind im Bereich HCI Leitfäden entstanden, die das Vorgehen in mehrere Stufen unterteilen. Ein bekanntes Beispiel ist „The LUCID Design Framework“ von der *Cognetics Corporation*. In jeder Entwicklungsstufe wird definiert, welche Voraussetzungen erfüllt sein müssen, um die jeweilige Stufe zu erreichen, was in der Stufe bearbeitet werden soll und welche Ergebnisse eine Stufe abschließen. LUCID beinhaltet eine exakte Entwicklungsstrategie vom ersten Prototypen bis zum *Roll Out* (Veröffentlichung der Software) der fertigen graphischen Oberfläche.
Auch bei kleinen Projekten, wie dieser Ausarbeitung, kann die Ausrichtung der Entwicklung an den LUCID Richtlinien hilfreich und zielführend sein. [Cog99]

### 2.2.4 Prototyping

Durch geeignetes *Prototyping* (Erstellung von Prototypen) können schon in frühen Phasen der Entwicklung erste Fehler erkannt und korrigierende Maßnahmen eingeleitet werden. Zu Beginn bietet es sich an, Prototypen zu kreieren, die ohne Funktionalität dem Anwender vorgeführt werden. Dies erlaubt es, vor der eigentlichen Umsetzung und Programmierung bereits Rückmeldungen vom Benutzer einzuholen und das Produkt bei Bedarf an die entsprechenden Anforderungen anzupassen. Der Anwender fühlt sich in den Entwicklungsprozess integriert und eine auf ihn zugeschnittene Software kann erstellt werden.
Für die Konzeption des Prototyps können unterschiedliche Techniken angewandt werden. Entwürfe und Skizzen lassen sich auch mit Papier und Bleistift erstellen. Solche *Storyboards* sind erste Visualisierungen von Konzepten oder Ideen auf Papier. Auch der Einsatz von *GUI Prototyping Tools* ist denkbar. Dies sind spezielle Programme zur Erstellung von Prototypen graphischer Oberflächen am Computer. Sie sind für weiter fortgeschrittene Prototypen geeignet, die am Bildschirm entwickelt werden.

### 2.2.5 Evaluierung der Benutzungsschnittstellen

Selbst wenn zur Erstellung der GUI alle Regeln und Prinzipien von HCI eingehalten wurden, garantiert dies noch keine „gute“ Lösung. Neben der notwendigen Evaluierung nach Beendigung des Projekts sind Tests im Verlauf der Entwicklung selbst wichtig. Dies ermöglicht ein frühes Korrigieren von Fehlern und erspart oft umfangreiche Verbesserungsmaßnahmen. [DFAB98]
Evaluierungen mit Benutzern können entweder unter Laborbedingungen oder im Feldversuch direkt an realen Projekten vorgenommen werden. Beide Vorgehen haben ihre Vor- und Nachteile.
Um die Usability einer graphischen Oberfläche zu bewerten, gibt es mehrere gängige Metho-

den, von denen zwei im Folgenden kurz vorgestellt werden.

Beim ***Cognitive Walkthrough*** wird eine Liste von Aufgaben erstellt, die mit dem zu testenden Programm erfüllt werden sollen. Eine Person, die den Test durchführt, arbeitet diese Aufgaben ab und soll dabei potentielle Usability Probleme aufspüren. Für diese Art der Evaluierung ist ein Prototyp der Software ausreichend, allerdings muss dieser einen möglichst hohen Detailgrad aufweisen. Im Verlauf der Evaluation müssen für jede ausgeführte Aktion die folgenden Fragen beantwortet werden:

- Hat die ausgeführte Aktion den vom Benutzer erwarteten Effekt?
- Ist die Schaltfläche, die der Benutzer für die nächste Aktion benötigt, sichtbar?
- Die Beantwortung der vorherigen Frage vorausgesetzt, weiß der Benutzer, dass diese Aktion die richtige ist, um die Aufgabe zu erfüllen?
- Hat der Benutzer die korrekte Aktion ausgeführt und versteht er die Rückmeldung, die er daraufhin erhält?

Beantwortet ein Tester eine Frage mit „nein“, muss dieser Bereich auf seine Usability überprüft werden und die graphische Oberfläche eventuell nachgebessert werden.

Bei der ***Heuristic Evaluation*** findet die Bewertung anhand von vorher festgelegten Gestaltungsprinzipien, den Heuristiken, statt. Mehrere Experten auf dem Gebiet Usability evaluieren die graphische Oberfläche gleichzeitig und tauschen im Anschluss ihre Ergebnisse aus. Die heuristische Evaluierung kann schon in frühen Phasen der Softwareentwicklung eingesetzt werden, da sie auf unfertigen Prototypen sowie auch auf dem fertigen Softwareprodukt ausgeführt werden kann. Im Folgenden werden die zehn Heuristiken nach *Jakob Nielsen* kurz vorgestellt.

1. Der Systemstatus sollte für den Benutzer immer sichtbar sein.
2. Die verwendete Sprache sollte für den Benutzer angemessen und verständlich sein.
3. Die Software soll für den Benutzer gut steuerbar sein.
4. Einhaltung von Konsistenz und Standards.
5. Fehlervermeidung.
6. Wiedererkennung von Elementen.
7. Flexibilität und Effizienz durch „Abkürzungen“ für erfahrene Nutzer.
8. Ästhetisches minimalistisches Design.
9. Verwendung geeigneter Fehlermeldungen.
10. Ausreichend Dokumentation und eine geeignete Hilfe Funktion.

Die Wahl der geeigneten Methode zur Evaluierung hängt beispielsweise von den entstehenden Kosten, der benötigten Zeit oder dem Entwicklungsstatus der graphischen Oberfläche ab. Die Entscheidung wird auch von der Art der zu entwickelten Software und dem Umfeld des späteren Einsatzes entscheidend mitbestimmt. [DFAB98]

# 3 State-Of-The-Art Analyse

In diesem Kapitel soll ein Blick auf aktuelle Entwicklungen im Bereich der Erstellung von graphischen Oberflächen geworfen werden. Im Mittelpunkt stehen dabei große GUI-Systeme und deren Guidelines zum Erstellen graphischer Oberflächen. Zusätzlich werden weitere Programme analysiert, die ähnliche Funktionen wie der Shader Viewer aufweisen. Es werden Stärken und Schwächen analysiert und dadurch Ideen und Anregungen abgeleitet, die später in die eigene Entwicklung einfließen können.

## 3.1 Guidelines aktueller GUI Systeme

Wird eine graphische Oberfläche unter der Maßgabe einer guten Integrationsfähigkeit in einem GUI System entwickelt, sollte geprüft werden, ob dieser Systemhersteller spezielle Guidelines oder Style Guides veröffentlicht und für Entwickler zu Verfügung gestellt hat. Einige große Anbieter von Betriebssystemen bieten solche Richtlinien an, in denen beschrieben wird, wie graphische Benutzeroberflächen aufgebaut werden sollen, um möglichst gut mit dem zugrunde liegenden Betriebssystem zu harmonieren. Werden die in den Guidelines empfohlenen Programme zur Entwicklung genutzt, kann das *Look and Feel* des GUI-Systems erfolgreich implementiert werden. Das Look and Feel beinhaltet standardisierte graphische Design Aspekte und eine konsistente Bedienbarkeit. In diesem Abschnitt sollen exemplarisch aktuelle Guidelines von zwei großen Betriebssystemherstellern etwas genauer betrachtet werden.

### 3.1.1 Apple

„Apple“ stellt für seine aktuelles GUI „Aqua“ die *Apple Human Interface Guidelines* und den *Apple Publications Style Guide* zur Verfügung. Zusätzlich gibt es Richtlinien zur Vermarktung von Produkten oder speziellen Produktnamen, die in dem *Apple Marketing Communications Style Guide* aufgeführt sind.
Der Apple Publications Style Guide enthält Regeln zur Darstellung von Text, der im Zusammenhang mit der Software verwendet wird. Hierunter fallen beispielsweise die Dokumentationen, Anleitungen oder innerhalb der Software dargestellte Texte. [App08b]
Im weiteren Verlauf dieses Abschnitts werden ausschließlich die Apple Human Interface Guidelines betrachtet, da diese den größten Einfluss auf die Entwicklung der graphischen Oberfläche haben. Im Folgenden werden ein paar Inhalte und ausgewählte Regeln vorgestellt.

*Cocoa*[6] und *Carbon*[7] unterstützen hier die Programmierung im Aqua Design. Wird die Darstellung und somit das Look and Feel des GUI-Systems weiterentwickelt, werden auch die

[6] Objektorientierte Programmierschnittstelle für das Betriebssystem Mac OS X von Apple.
[7] Von Apple entwickelte Sammlung von Programmierschnittstellen für Mac-Betriebssysteme.

in der alten Darstellungsart programmierten Oberflächen in das neue Design überführt. Als Prototyping Tool stellt Apple den „Interface Builder“ zur Verfügung, dessen Entwürfe später mit Cocoa oder Carbon weiter verarbeitet werden können. Vom Apple GUI System reservierte Tastenkombinationen dürfen nicht mit eigenen Funktionen überschrieben werden. Sie bleiben dem Betriebssystem vorbehalten. In den Guidelines von Apple wird empfohlen, graphische Oberflächen, neben der Verwendung eines Pointing Devices, auch komplett mit der Tastatur steuern zu können. Dies bietet sich vor allem bei Laptops an, deren *Touchpads*[8] nur bedingt für eine genaue Steuerung geeignet sind.
Häufig verwendete Elemente in graphischen Oberflächen werden so positioniert, dass sie möglichst schnell erreichbar sind. Selten verwendete Elemente rücken in den Hintergrund oder in Untermenüs. Auch zum Layout von Menüs werden Angaben gemacht. Beispielsweise werden untereinander angeordnete Steuerelemente linksbündig ausgerichtet, gleichartige Elemente haben die identische Breite und das Gesamtlayout ist zentriert und symmetrisch.
Bei der Analyse des Benutzerprofils wird oft auf dessen *Mental Model* eingegangen. Hierbei handelt es sich um Annahmen über die wahrscheinliche Strategie, mit der ein Anwender an die Lösung einer Aufgabe herangehen wird. Entspricht die Steuerung der graphischen Oberfläche genau diesem Modell, sollte sie optimal auf den Benutzer zugeschnitten sein.
*Forgiveness* beschreibt die Fehlertoleranz einer Software. Durch die Möglichkeit, wieder in den Zustand vor dem unterlaufenen Fehler zurückzukehren, kann der Benutzer wesentlich unbeschwerter mit der Software arbeiten.
Dies stellt nur eine kleine Auswahl von Themen dar, die in den Apple Guidelines detailliert betrachtet werden. [App08a] Viele Entscheidungen werden dem Benutzer abgenommen und nur selten kann dieser aufgrund von optischen Präferenzen zwischen alternativen Darstellungen wählen.

### 3.1.2 Microsoft

Die von „Microsoft“ zur Verfügung gestellten Guidelines nennen sich *Windows User Experience Interaction Guidelines*, abgekürzt „UX Guide“. Sie beziehen sich hauptsächlich auf das Betriebssystem „Vista“ und dessen graphische Oberfläche „Aero“. Als Programmierschnittstelle wird die „Windows Presentation Foundation“ angeboten. Dabei handelt es sich um ein graphisches Programmiergerüst (Framework), auch unter dem Namen „Avalon“ bekannt, welches Teil des „.NET Frameworks 3.0“ ist. Diese Software Plattform besteht aus Laufzeitumgebung, Klassenbibliothek und Dienstprogrammen, mit deren Hilfe graphische Oberflächen mit dem Vista Look and Feel erstellt werden können.
Im Folgenden werden ein paar ausgewählte Aspekte der Guidelines vorgestellt. Es sind viele Aspekte aus der HCI enthalten, die bereits im Kapitel 2 vorgestellt wurden. Elemente, die

[8] Tastfeld, Eingabegerät welches aus einer berührungsempfindlichen Fläche besteht.

bei Apple in einem extra Publications Style Guide ausgelagert waren, sind bei Microsoft mit im UX Guide enthalten. So werden unter anderem auch Regeln zum „Umgangston“ mit dem Benutzer gemacht. Es wird darauf Wert gelegt, dass dieser nicht zu „arrogant“ oder „bestimmend“ wirkt. Besonders bei Fehlermeldungen, soll der Benutzer nicht direkt angesprochen werden, um ihm nicht den Eindruck zu vermitteln, er selbst habe den Fehler verschuldet. Inkonsistenten Formulierungen, die im Computerumfeld oft nicht klar definiert sind, wird in den Guidelines vorgebeugt. Beispielsweise ist die Mehrzahl von „Mouse“ (Maus) nicht „Mice“ oder „Mouses“ sondern „Mouse Devices“.
Zur Verwendung von *Tastatur Shortcuts*[9] gibt es eine Reihe unterschiedlicher Regeln. Kombinationen sollen gewählt werden, die einfach mit den Fingern zu erreichen sind. Die Tasten sollten daher nicht zu weit auseinander liegen und deren Kombinationen dürfen nicht mit reservierten systemspezifischen Shortcuts kollidieren. Nach Möglichkeit sollte der Shortcut den Anfangsbuchstaben des Befehls enthalten, da sich dies beim Benutzer leichter einprägt (z.B. STRG + C für Copy).
In den Guidelines werden unterschiedliche Fenstertypen mit variierenden Eigenschaften vorgestellt. Jedes dieser Fenster ist für eine spezielle Situation geeignet. Die Fenster unterscheiden sich beispielsweise darin, von wem sie aufgerufen werden, ob sie in der Startleiste angezeigt werden oder in ihrer Beziehung zu anderen Fenstern. Um die Software für möglichst viele Benutzer zugänglich zu machen, werden Hinweise auf maximale und minimal Bildschirmauflösungen gegeben.
Menüs und vor allem Fehlermeldungen sollten möglichst nah an der sie aufrufenden Stelle platziert werden. Somit kann die Struktur der Menüführung vom Benutzer besser nachvollzogen werden.
Auch dies ist wieder nur ein kleiner Auszug aus den vielen Themenbereichen, die in den Guidelines von Microsoft angesprochen werden. Die Beschreibung der aufgeführten Regeln ist sehr detailliert und mit vielen Schaubildern, direkt aus der Aero GUI, versehen. [Mic08]

### 3.1.3 Vor- und Nachteile, Fazit

Am Ende dieses Abschnitts über Guidelines von Herstellern großer GUI-Systeme sollen die Vor- und Nachteile gegenüber gestellt werden, die eine Entwicklung entsprechend dieser Guidelines mit sich bringt.

**Vorteile:**
Haben sich Benutzer stark an das Look and Feel eines Betriebssystems gewöhnt, fällt es ihnen oft leichter, sich mit einer Software auseinanderzusetzen, die ein ähnliches Aussehen besitzt und bekannten Prozeduren folgt. Sie lernen schneller mit der Software umzugehen, da viele Eigenschaften und Befehle bereits bekannt sind.

[9] Tastenkombinationen, Tastenkürzel

Wird das Betriebssystem GUI weiterentwickelt, also ein neues Oberflächendesign eingeführt, kann das eigene Softwareprodukt automatisch mit in diese Version aufgenommen und angepasst werden. Die entwickelte graphische Oberfläche wirkt auch in das neue GUI-System integriert und aktuell.
Ein weiterer Vorteil der Guidelines ist, dass deren Hersteller viel Erfahrung, Kompetenz und Entwicklungsarbeit in die Erstellung ihres GUI-Systems investiert haben. Wendet man die Regeln der Guidelines an, kann man sich diese Entwicklungsleistungen zu Nutzen machen. Das Misserfolgsrisiko einer neu entwickelten Software, das auf einer graphischen Oberfläche mit schlechter Usability beruht, wird minimiert. Der zeitliche Aufwand beim Programmieren einer graphischen Oberfläche kann erheblich reduziert werden, wenn vorgefertigte Steuerelemente verwendet werden und diese lediglich in Menüs miteinander verknüpft werden müssen. Auch wirtschaftliche Vorteile können entstehen, da es meist kostengünstiger ist, bestehende Guidelines zu verwenden, anstatt ein neu entwickeltes eigenes Design anzuwenden.

**Nachteile:**
Mit der Wahl eines GUI Systems legt man sich auf ein Design fest. Die Entwicklung ist somit an spezielle Schnittstellen und Entwicklungssoftware gebunden. Flexibilität geht verloren. Ein gewisses Maß an Entscheidungsfreiraum wird aus der Hand gegeben.
Richtlinien können zwar unterschiedlich strikt umgesetzt und befolgt werden, allerdings verliert die Software stark an Individualität. Dies kommt allein schon dadurch zustande, dass vorgegebene Steuerelemente - unabhängig davon wie sie auf dem Bildschirm erscheinen oder verknüpft werden - immer ein ähnliches optisches Ergebnis liefern.
Legt man sich auf das Design eines bestimmten GUI-Systems fest, wird auch zu einem gewissen Teil das Image des jeweiligen Herstellers mit übernommen. Dies kann sich natürlich sowohl positiv als auch negativ auf ein Produkt auswirken. Weiterhin besteht die Gefahr, sich nicht den kompletten Markt für eine Software erschließen zu können, wenn man sich auf bestimmte Guidelines festlegt. Diesem Effekt kann entgegen gewirkt werden, indem verschiedene Versionen einer Software veröffentlicht werden, die auf verschiedene GUI-Systeme zugeschnitten sind und die jeweiligen Guidelines implementieren. Dies führt natürlich zu einem größeren Entwicklungsaufwand und ist mit höheren Kosten verbunden.

**Fazit:**
Es ist zu überlegen, ob eine selbst entwickelte graphische Oberfläche mit eigenem Design nicht doch die bessere Alternative darstellt. Zusätzlich kann mit einer selbst entwickelten graphischen Oberfläche ein Wiedererkennungswert beim Benutzer geschaffen werden. Ist das eigene Produkt gut durchdacht und beinhaltet innovative Ideen bei der Realisierung der graphischen Oberfläche, können Kunden besser an das eigene Produkt gebunden werden. Insgesamt gesehen lässt sich aber keine allgemeingültige Aussage für oder gegen die Verwendung von Guidelines treffen. Die Entscheidung ist situationsabhängig.

## 3.2 Shader Viewer anderer Hersteller

Innerhalb der *State-Of-The-Art* Analyse werden auch mit dem Shader Viewer verwandte Produkte genauer betrachtet. Es gibt einige Hersteller, die Software auf dem Markt anbieten, mit der Shader Effekte visualisiert werden können. Der Schwerpunkt der Betrachtung wird auf Usability und Design der entsprechenden Software gelegt. Diese Kriterien sollen anhand der vorgestellten HCI Richtlinien analysiert werden. Auf eine weiterreichende Betrachtung der Software, beispielsweise die korrekte Darstellung der Shader Effekte betreffend, soll hier verzichtet werden.

In den nächsten Abschnitten werden exemplarisch zwei Shader vorgestellt. Dabei handelt es sich um den „FX Composer“ der Firma „NVIDIA“ und um den „RenderMonkey“ der Firma „AMD/ATI“. Beide stehen kostenfrei im Internet zum Download zur Verfügung[10].

Bei den graphischen Oberflächen der beiden Programme handelt es sich um *Multiple Document Interfaces* (MDI). Das sind Oberflächen, bei denen in einem Programm mehrere Unterfenster (Inner Frames) angezeigt werden können. Diese Unterfenster können frei platziert und in ihrer Größe verändert werden. Bei beiden Programmen sind die Unterfenster als Mischformen aus *Single Document Interfaces* und *Tabbed Document Interfaces* angelegt. Im Single Document Interface wird jedes Dokument, also jeder Inhalt, in einem eigenen Unterfenster dargestellt. Beim Tabbed Document Interface werden Unterfenster in Registerkarten organisiert. Dadurch können mehrere unterschiedliche Inhalte platzsparend in einem Unterfenster dargestellt werden. Allerdings sind die Inhalte nicht zeitgleich sichtbar und deshalb ist diese Methode nicht immer sinnvoll.

Zur Analyse der graphischen Oberfläche dieser beiden Programme werden ähnlich wie beim *Cognitive Walkthrough*, der im Abschnitt 2.2.5 bereits vorgestellt wurde, ein paar vorher definierte Aufgaben mit der Software abgearbeitet. Dabei werden Objekte angelegt, von Lichtquellen beleuchtet und mit Texturen versehen. Neben den Erkenntnissen, die beim Ausführen der Aufgaben erlangt werden, soll auch ein allgemeiner Eindruck der graphischen Oberfläche dokumentiert werden. Eine Bewertung des Designs und der Aspekte, die besonders positiv oder negativ auffallen, wird ebenfalls vorgenommen. Falls möglich werden diese Aspekte durch Screenshots visualisiert. Zum Schluss wird ein Fazit gezogen, das die Ergebnisse kurz zusammenfasst.

[10] FX Composer: `http://developer.nvidia.com/object/fx_composer_home.html`
zuletzt besucht am 14.03.2009
RenderMonkey: `http://developer.amd.com/gpu/rendermonkey/Pages/default.aspx`
zuletzt besucht am 14.03.2009

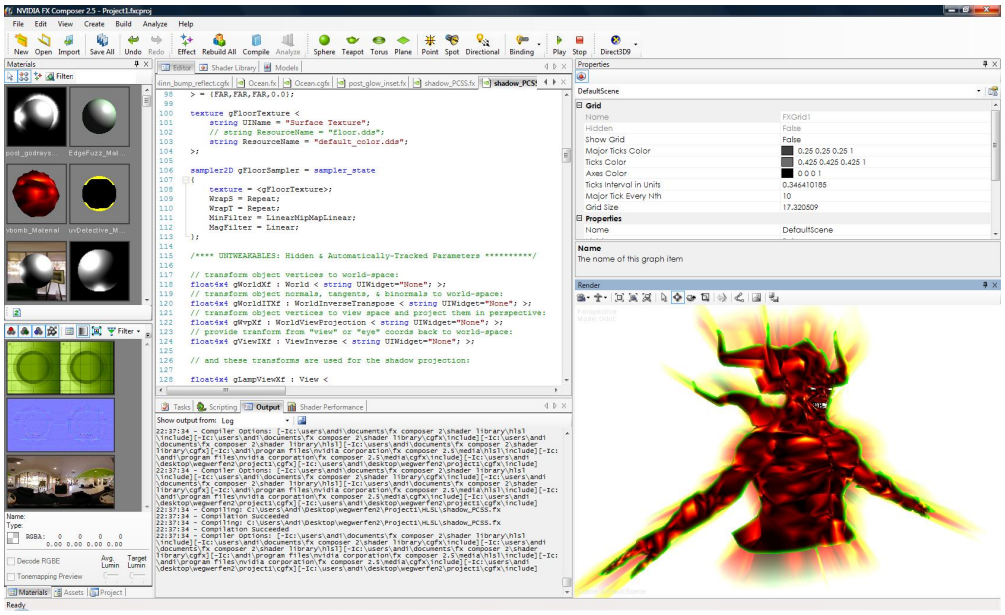

Abbildung 3: Screenshot FX Composer

### 3.2.1 FX Composer

Zum Test liegt die aktuelle Version 2.5 vom Juli 2008 des FX Composers vor. Es handelt sich hierbei um eine Entwicklungssoftware für Shader der Firma NVIDIA.

**Erster Eindruck**

Auf den ersten Blick wirkt der FX-Composer sehr komplex und unübersichtlich. Die graphische Oberfläche unterteilt sich in mehrere Unterfenster, welche teilweise in Registerkarten organisiert sind. Der Benutzer muss sich erst einen Überblick verschaffen, welche Informationen in welchem Unterfenster dargestellt werden und wie sie auf dem Bildschirm angeordnet sind.
Ein konstantes Element in der graphischen Oberfläche ist das Hauptmenü, das auf dem oberen Teil des Bildschirms zu sehen ist. In diesem können Grundeinstellungen des Programms vorgenommen werden. Die Icons im Hauptmenü sind relativ groß und dadurch gut zu erkennen. Durch die graphischen Icons und die damit verbundenen *Metaphern*[11] ist meist direkt auf die Funktion der Schaltflächen zu schließen. Dies wird durch einen kurzen Beschreibungstext innerhalb des Icons noch verstärkt. Die Standardbildschirmaufteilung und das Hauptmenü sind in Abbildung 3 zu sehen.

Um die Einarbeitung zu beschleunigen, sind diverse Beispielprogramme verfügbar, welche zu individuellen Projekten weiterentwickelt werden können. Zusätzlich wird direkt im Startmenü ein „Quick Tutorial“ angeboten. Dabei handelt es sich um eine kurze Anleitung / Übung, um die grundlegenden Funktionen des Programms schnell zu erlernen.

[11] Symbol, das sinnbildlich für etwas anderes steht.

### Arbeiten mit dem Programm

Es fällt auf, dass viele Benutzeraktionen durch *Drag and Drop*[12] mit der Maus ausgeführt werden können. Beispielsweise kann ein Objekt texturiert werden, indem mit der Maus die Textur aus dem Textur-Unterfenster direkt in das Unterfenster mit dem gerenderten Objekt gezogen wird. Dies gilt auch für Effekte, die im Effekt-Unterfenster untergebracht sind.

### Positive Aspekte

Besonders positiv ist die stark individualisierbare graphische Oberfläche aufgefallen.

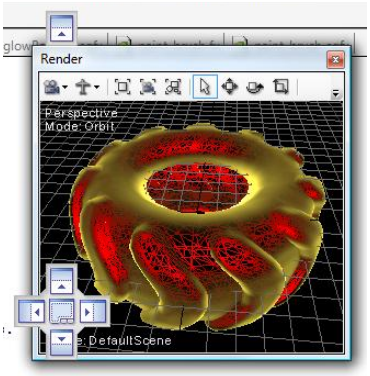


Abbildung 4: FX Composer Unterfenster Positionierung

Unterfenster können frei innerhalb der graphischen Oberfläche bewegt werden und in ihrer Größe verändert werden. Die Unterfenster lassen sich aus ihrer Verankerung lösen, um an einer anderen Stelle des Bildschirms eine neue feste Position zu erhalten.

Abbildung 4 zeigt, wie dem Benutzer durch ein „Positionierungskreuz“ visualisiert wird, an welcher Stelle ein Unterfenster neu positioniert werden kann. Zur besseren Übersichtlichkeit können nicht verwendete Unterfenster auch komplett geschlossen werden.

Ein weiterer Aspekt zur Steigerung der Usability ist die Auswahl von unterschiedlichen Layouts. Es stehen einige vordefinierte Layouts zur Verfügung. Es können aber auch Eigene erstellt, gespeichert und geladen werden. Ein Layout bestimmt, welche Unterfenster auf dem Bildschirm angezeigt werden und wie diese angeordnet sind.
Die vom System vordefinierten Layouts sind:

- *Artist*: Das Unterfenster mit dem gerenderten Bild steht im Vordergrund und nimmt den meisten Platz ein. Es wird besonderer Fokus auf das Ergebnis des Renderns und den graphischen Effekt, den der Shader erzeugt, gelegt.
- *Authoring*: Das *Properties-Unterfenster* zur Speicherung von Shader Einstellungen befindet sich an exponierter Stelle. Ansonsten ist dieses Layout dem *Default* Layout sehr ähnlich.
- *Default*: Allgemeines Layout, das nicht auf eine Aufgabe spezialisiert ist.
- *Power User*: Die Unterfenster mit dem Programmcode des Shaders und mit dem gerenderten Ergebnis stehen im Vordergrund. Die restlichen Fenster sind als kleine Symbole am Rand dargestellt. Bei Bedarf schieben sie sich in den sichtbaren Bereich des Bildschirms.
- *Tuning*: Layout zur Feinabstimmung des Shaders. Es ist das einzige Layout, in dem das

[12] Bedienungsmethode graphischer Oberflächen, bei der graphische Elemente mittels eines Zeigegerätes bewegt werden.

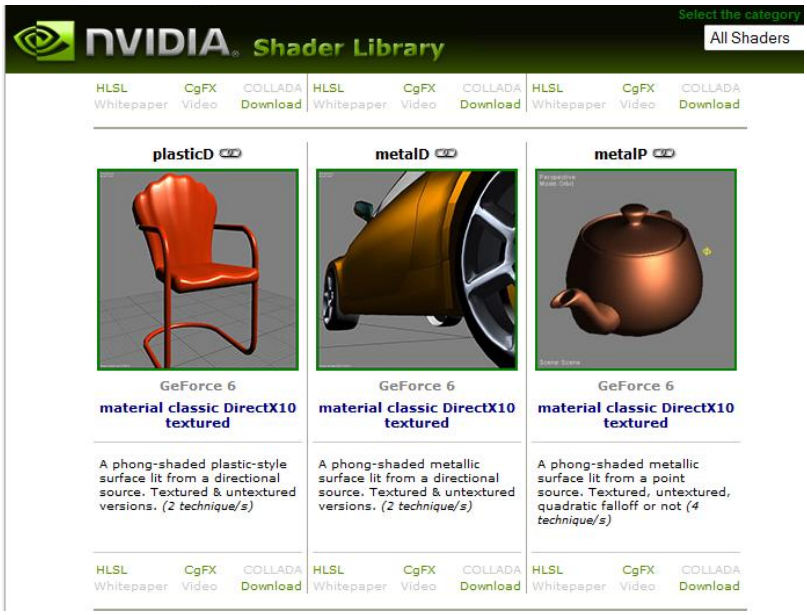


Abbildung 5: FX Composer Shader Library

*Performance-Unterfenster* eingeblendet ist. In diesem kann die Laufzeit des Shaders mit unterschiedlichen Graphikkarten und Treibern analysiert werden.

Die unterschiedlichen Layouts sind für spezifische Arbeitsschritte bei der Shader Entwicklung geeignet. Die wichtigen Unterfenster sind meist zentriert dargestellt und nehmen den größten Teil des Bildschirm ein. Zu Beginn der Entwicklung können im Default Layout alle benötigten Ressourcen geladen werden. In der nächsten Entwicklungsphase wird im Power User Layout die Shader Programmierung durchgeführt und zum Schluss werden im Tuning Layout noch letzte Änderungen zur Laufzeitverbesserung vorgenommen.

Wurden viele Unterfenster in ihrer Darstellung verändert, kann der Benutzer leicht den Überblick verlieren. Hierfür gibt es einen *Reset Layout* Menüpunkt, der das Layout auf die Standardeinstellungen zurücksetzt.

Positiv ist auch die *NVIDIA Shader Library* zu erwähnen. Hier können Beispielshader in das Programm geladen werden. Durch große und detailreiche Abbildungen wird dem Benutzer visualisiert, welche Shadereffekte der Beispielshader realisiert. Auch wenn die Fachbegriffe des jeweiligen Shadereffektes nicht bekannt sind, kann der Benutzer eine Auswahl treffen, die seinen Vorstellungen entspricht. In Abbildung 5 ist ein Ausschnitt aus der Shader Library zu sehen.

**Negative Aspekte**

Da nicht alle Unterfenster auf dem Bildschirm platziert werden können, sind einige Positionen doppelt oder mehrfach belegt. Die Unterfenster werden mit Reitern versehen, um die einzelnen Registerkarten aufzurufen. Durch die überladen wirkende Oberfläche des FX Composers

können Informationen leicht übersehen werden.
Die Ausrichtung der Kamera und des Objektes ist im Unterfenster, in dem das gerenderte Bild zu sehen ist, nicht intuitiv bedienbar.
Die Unübersichtlichkeit durch eine hohe Anzahl an Details und unterschiedlicher Unterfenster wird bei kleinen Bildschirmen oder niedrigen Bildschirmauflösungen noch problematischer. Der Bildschirm wirkt überladen, dem Benutzer werden zu viele Informationen präsentiert. Der Anwender dürfte schnell überfordert sein.

**Fazit**
Das Programm wirkt durch seine Vielzahl an Informationen und Auswahlmöglichkeiten sehr mächtig und vielseitig. Ob die vielfältigen Informationen und Auswahlmöglichkeiten wirklich so nützlich und sinnvoll sind, wurde allerdings mit den hier durchgeführten Tests nicht geprüft. Die überladene, unübersichtliche Oberfläche ist problematisch.
Es werden zwar Möglichkeiten geboten, die graphische Oberfläche individuell zu gestalten und auf Informationen zu verzichten, allerdings erfordert dies eine gewisse Einarbeitung in die Software.

### 3.2.2 RenderMonkey

Bei dieser Betrachtung wird die RenderMonkey Version 1.82 vom 18.12.2008 verwendet. Es ist eine Shader Entwicklungsumgebung der Firma AMD/ATI. Der Rendermonkey richtet sich sowohl an Programmierer wie auch an Designer im Bereich Shaderentwicklung.

**Erster Eindruck**
Auf den ersten Blick weist der RenderMonkey kein sehr elegantes und ästhetisches Design auf. Vor allem nach der Verwendung des FX Composers wirkt seine graphische Oberfläche etwas plump und veraltet. Dies wird jedoch durch eine wesentlich bessere Übersichtlichkeit kompensiert. Ähnlich wie beim Produkt aus dem Hause NVIDIA befindet sich auf dem oberen Teil des Bildschirms das Hauptmenü und darunter liegend mehrere Unterfenster. Auf der linken Seite wird der Workspace als Baumstruktur dargestellt. Je nach Bedarf können Unterfenster hinzugefügt oder entfernt werden. Der Benutzer kann die Unterfenster an unterschiedliche Stellen des Bildschirms verschieben und deren Größe verändern. Somit kann das Programm den Ansprüchen des Benutzers entsprechend angepasst werden. Abbildung 6 zeigt einen Screenshot des Programms, in dem eine Vielzahl der möglichen Unterfenster eingeblendet sind.
Beim Speichern eines Projekts werden neben den Daten zum Shader auch die geöffneten Unterfenster und ihre Positionen gespeichert. Dies erleichtert es dem Benutzer, nach einer Unterbrechung an einem Projekt weiter zu arbeiten. Durch die relativ geringe Zahl an Unterfenstern können fast alle Unterfenster über die Icons im Hauptmenu hinzu geschaltet oder ausgeblendet werden. Diese Icons sind leider etwas klein ausgefallen und somit nur schwer zu

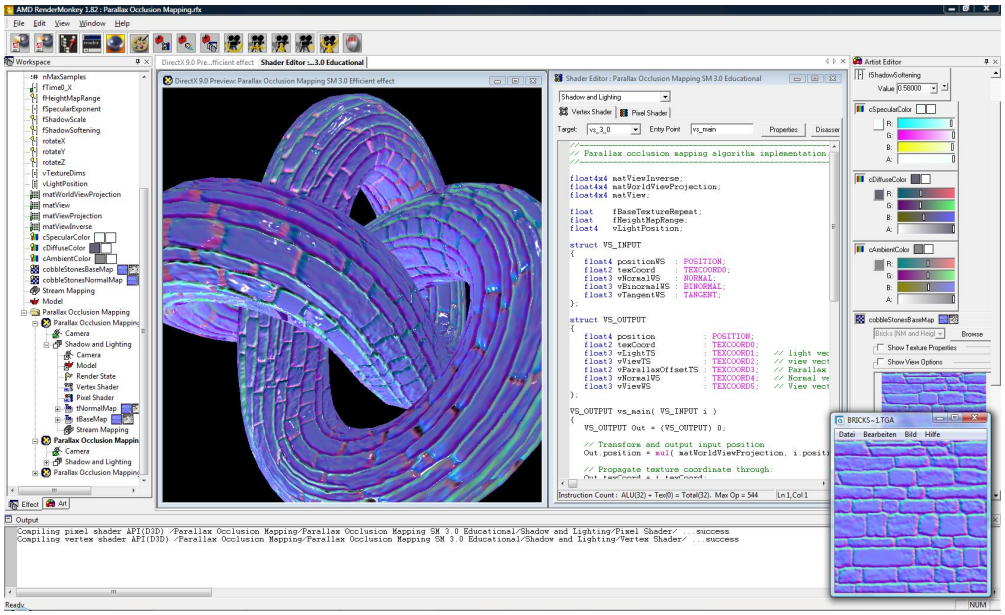

Abbildung 6: Screenshot RenderMonkey

Abbildung 7: RenderMonkey Icons

erkennen. Man versteht meist nicht durch bloßes betrachten, welche Funktion sich hinter den Icons verbirgt. Hier hilft oft nur der *Tooltip-Text* weiter, der erscheint, wenn die Maus eine Zeit lang auf dem Icon nicht bewegt wird. Abbildung 7 zeigt eine Übersicht über die verfügbaren Icons des Hauptmenüs.
In der Installation sind mehrere Beispielprojekte enthalten.

**Arbeiten mit dem Programm**

Die Workspace Baumstruktur stellt das zentrale Unterfenster des RenderMonkeys dar. Hier werden alle Projekte strukturiert dargestellt und alle Materialen und Ressourcen ausgewählt und hinzugefügt. Verschiedene Render-Durchläufe (Renderpasses) und alle dazu benötigten Objekte und Parameter können hier angelegt werden.

**Positive Aspekte**

Innerhalb des Unterfensters, in dem das gerenderte Objekt dargestellt wird, ist für den Benutzer gut zu erkennen, auf welche Art das Objekt oder die Kamera gerade bewegt werden kann. Der Bewegungsmodus der Maus wird ihm über das markierte Icon im Hauptmenü angezeigt. Das Programm wirkt übersichtlich und gut strukturiert. Im Workspace-Unterfenster können die wichtigsten Informationen zum Projekt gespeichert und modifiziert werden. Innerhalb der Baumstruktur des Workspaces werden Drag and Drop Operationen unterstützt. Knoten kön-

nen dadurch einfach an eine andere Stelle im Baum kopiert werden. Es ist auch möglich, Dateien wie beispielsweise Texturen oder Objekte aus dem Datei System per Drag and Drop in den Workspace Baum zu integrieren. Hierbei wird direkt geprüft, ob die vorgenommen Operation innerhalb des Baumes zulässig ist und dem Benutzer direkt Rückmeldung gegeben, ob sein gewähltes Objekt an die entsprechende Stelle kopiert werden darf. Nach diesen Operationen wird der Workspace-Baum automatisch neu organisiert.

Um den Baum im Workspace-Unterfenster etwas übersichtlicher zu gestalten, können Notizen zur Dokumentation eingefügt werden, wie in Abbildung 8 zu sehen ist. Dies sind spezielle Knoten, die allgemeine Informationen enthalten können und den Baum somit leserlicher machen.

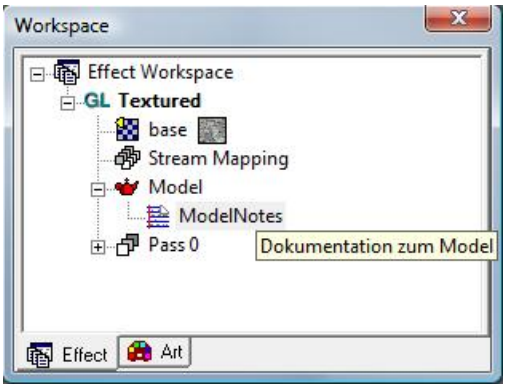


Abbildung 8: RenderMonkey Baumstruktur

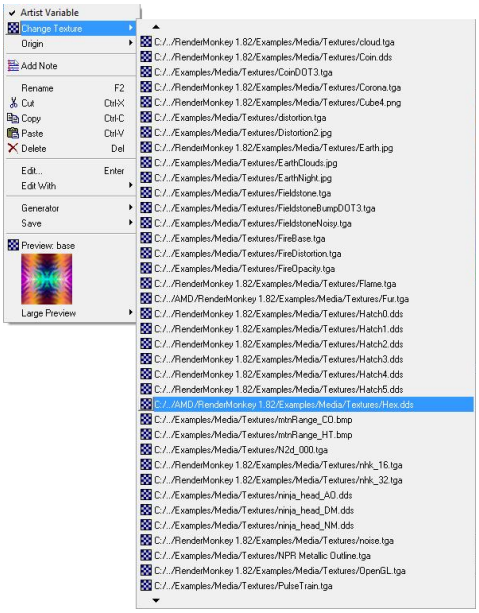


Abbildung 9: RenderMonkey Listendarstellung

**Negative Aspekte**

Es lassen sich zwar sehr viele Operationen im Workspace Baum effizient durchführen, allerdings wird dieser Baum bei großen Projekten selbst schnell unübersichtlich. Ein weiterer Nachteil sind die riesigen Listen, die bei der Auswahl von Texturen, Objekten oder anderen Elementen entstehen können. In Abbildung 9 ist beispielhaft eine dieser schlecht navigierbaren Listen abgebildet.

Auch andere Menüs wirken optisch missglückt. In Abbildung 10 ist als Beispiel ein Kamera-Menü zu sehen. Die Darstellung wirkt zu dicht gedrängt und dadurch unübersichtlich. Etwas mehr Abstand zwischen den Elementen oder eine Gruppierung zwischen den Einstellungen hätte diesem Effekt entgegenwirken können.

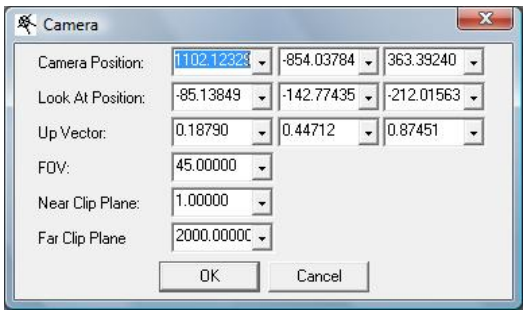


Abbildung 10: RenderMonkey Kamera Menü

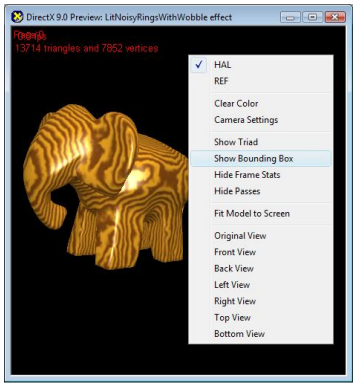


Abbildung 11: RenderMonkey Kontextmenü

Unglücklich ist auch die Umsetzung des in Abbildung 11 gezeigten Kontextmenüs. Nach Auswahl einer Option wechselt hier das Programm zwischen der Anzeige von *Show* und *Hide*, d.h. zwischen Anzeigen und Ausblenden einer Option. Durch farbliches Hervorheben oder Markieren der angezeigten Optionen könnte eine größere Übersichtlichkeit und Lesbarkeit erzielt werden.
Weiterhin ist in der oberen linken Ecke von Abbildung 11 zu sehen, dass bei gleichzeitigem Anzeigen von *Frame State*[13] und *Passes*[14], diese übereinander dargestellt werden und nicht mehr lesbar sind.

**Fazit**

Was das Design der graphischen Oberfläche anbetrifft, macht der RenderMonkey auf den ersten Blick einen nicht sehr innovativen Eindruck. Dies bestätigt sich auch bei der weiteren Verwenden der Software. Viele kleinere Mängel bei der Gestaltung der Menüs fallen auf.
Allerdings ist der RenderMonkey wesentlich übersichtlicher als der FX Composer. Die graphische Oberfläche des RenderMonkey besitzt weniger Einstellungsmöglichkeiten, was allerdings nicht negativ auffällt, da die Menüs schlichter und einfacher gehalten sind.
Durch die Baumstruktur des Workspaces lassen sich unkompliziert neue Elemente in das Projekte einbinden. Werden Projekte sehr groß, besteht allerdings die Gefahr, dass die Baumstruktur ihre Übersichtlichkeit verliert.

[13] *Frames per Second*, Zahl der Einzelbilder pro Sekunde.
[14] Render Durchlauf

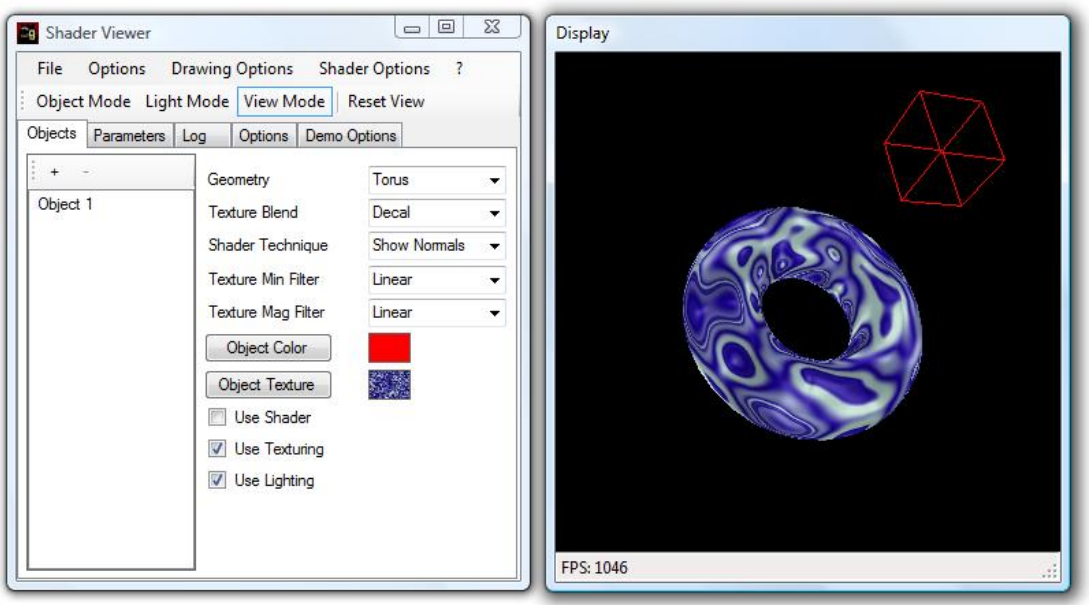


Abbildung 12: Screenshot Shader Viewer

## 3.3 Shader Viewer

Am Ende dieses Kapitels wird kurz der Shader Viewer mit seiner aktuellen graphischen Benutzeroberfläche vorgestellt. Dieses Programm lässt sich nicht unter den gleichen Voraussetzungen mit dem FX Composer und dem RenderMonkey vergleichen. Es wurde Ende 2008 im Rahmen einer Diplomarbeit von Daniel Schiffner erstellt. Hieraus wird schon deutlich, dass es nicht den Funktionsumfang wie die Programme von AMD/ATI und NVIDIA haben kann, welche zeitaufwendiger von vielen Entwicklern erstellt wurden. Der Shader Viewer soll die Effekte von Shadern visualisieren und deren Parameter anpassen können. Hierzu werden keine Shader Programme in der Software selbst erstellt, sondern diese werden importiert und dargestellt. Die graphische Oberfläche wurde mit Standard .NET Komponenten kreiert.

### Erster Eindruck

Dieses Programm verfolgt nicht den MDI Ansatz, sondern setzt sich aus mehreren Fenstern zusammen. Abbildung 12 zeigt einen Screenshot des Shader Viewers. Das auf der linken Seite dargestellte Fenster beinhaltet die graphische Oberfläche zur Steuerung und Eingabe von Parametern. Im rechten Fenster wird die gerenderte Ausgabe des Shaders visualisiert.

### Arbeiten mit dem Programm

Das Fenster zur Eingabe der Parameter ist in mehrere Registerkarten unterteilt. Dadurch wird eine thematische Gruppierung der Oberfläche erreicht.
Im Hauptmenü am oberen Rand des Bildschirms kann zwischen drei Modi gewählt werden:

*Object Mode*, *Light Mode* und *View Mode*[15]. Diese Modi geben an, ob das Objekt, die Lichtquelle oder die Kamera im Ausgabefenster gerade bewegt werden kann.

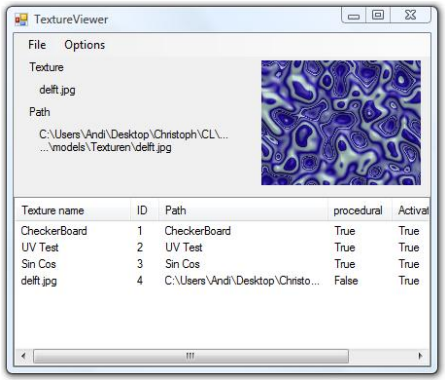


Abbildung 13: Shader Viewer Texturauswahlmenü

In Abbildung 13 ist das Auswahlmenü für Texturen zu sehen, welches in einem extra Fenster dargestellt wird.

**Positive Aspekte**

Durch die Auswahl der unterschiedlichen Modi im Hauptmenü hat der Benutzer ständig im Blickfeld, wie das betrachtete Objekt gerade modifiziert werden kann. Durch eine *Reset View* Funktion, können die unterschiedlichen Modi einzeln auf ihren Standardwert zurückgesetzt werden.

Über einen *Demo-Modus* kann eine Bewegung innerhalb der Szene durch Speichern einer Bildsequenz dargestellt werden.

**Negative Aspekte**

Einige aufgerufene Fenster müssen erst vom Benutzer geschlossen werden, bevor er mit dem Programm weiterarbeiten kann. Hierbei handelt es sich nicht um kritische oder akut relevante Informationen, die umgehend vom Benutzer verarbeitet werden müssen, sondern beispielsweise um das *Show Profile* Fenster, dass Informationen über den aktuellen Rendervorgang enthält. Da die Programmteile in unterschiedlichen Fenstern dargestellt werden, wirkt das Programm nicht wie eine Einheit. Das Design ist zwar funktional, hinterlässt durch die Verwendung von Standardkomponenten jedoch keinen sehr innovativen und einprägsamen Eindruck.

**Fazit**

Der Shader Viewer ist ein Programm zur Darstellung von Shader Effekten, dessen graphische Oberfläche durch die vielen einzelnen Fenster uneinheitlich wirkt.

[15] Modell-, Licht- und Betrachtungsmodus

# 4 Konzepte

In diesem Kapitel werden konkrete Vorstellungen zur Gestaltung und zum Design der graphischen Oberfläche des Shader Viewers entwickelt und näher beschrieben. Aspekte aus den Kapiteln Grundlagen und State-Of-The-Art Analyse lassen sich integrieren und fließen maßgeblich in die Entwicklung mit ein. Die Konzepte werden anhand von Skizzen und Prototypen visualisiert, um dem Leser zu veranschaulichen, welche Entscheidungen getroffen wurden und in die Umsetzung eingeflossen sind.

## 4.1 Analyse des Umfelds

Bevor erste Skizzen oder Prototypen erstellt werden können, müssen möglichst genaue Informationen über das spätere Einsatzgebiet der Software und die jeweiligen Benutzer gesammelt und ausgewertet werden. Der Shader Viewer wird hauptsächlich an der „Professur für Graphische Datenverarbeitung am Fachbereich Informatik und Mathematik der Goethe-Universität“ verwendet, an dem er auch entwickelt wurde. Hier wird er meist von Mitarbeitern benutzt, die mit der Materie Shader Programmierung vertraut sind. Auch Anwender aus dem künstlerischen Bereich, die Erfahrungen im Umgang mit Gestaltung und Farben besitzen, werden später mit dem Programm arbeiten. Allerdings sind auch Einsatzgebiete für Lehrzwecke denkbar. Deshalb sollte der Shader Viewer nicht nur auf erfahrene, fachkundige Anwender ausgelegt sein. Dies ist auch insofern generell von Vorteil, wenn der Benutzerkreis der Software später erweitert werden sollte. Andererseits sind Shader ein Themengebiet, mit dem sich meist nur Anwender auseinandersetzen, die im Umgang mit Computerprogrammen erfahren sind.

## 4.2 Prototyping

Für die ersten Gestaltungsideen der graphischen Oberfläche wurden Skizzen mit Papier und Bleistift angefertigt. Hierbei wurde zunächst ein Gestaltungsraster entworfen, das lediglich angibt, welche Position die verschiedenen Elemente der graphischen Oberfläche später auf dem Bildschirm einnehmen sollen [Thi00].

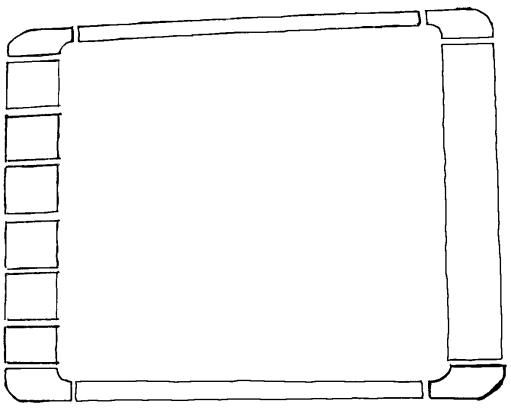

Abbildung 14: Gestaltungsraster

Ein solches Raster ist im linken Teil von Abbildung 14 zu sehen. Wie man sieht, wird die graphische Oberfläche aus einem Rahmen bestehen, der sich um die Ausgabe des Shader Viewers legt. Der Gefahr, durch zu viele Elemente die graphische Oberfläche zu überladen, konnte entgegengewirkt werden, in dem die Menüs schlank und schlicht gehalten wurden. Um den Benutzer nicht zu überfordern, sind viele Details in Untermenüs zusammengefasst, die nur bei Bedarf sichtbar sind. Es entsteht somit eine thematische Gliederung. Weitere handschriftliche Skizzen folgten, die erste Details zum visuellen Design der Oberfläche enthielten.

In Abbildung 15 ist in der rechten Hälfte eine erste farbige Skizze abgebildet, welche die obere linke Ecke der späteren Oberfläche darstellt. Darauf ist exemplarisch ein Farbmenü abgebildet. Bei diesem sehr frühen Entwurf, ist davon auszugehen, dass Elemente dieses Entwurfs bereits nach ersten Evaluierungen verändert werden müssen. Mit der abgebildeten Skizze soll lediglich ein erster Eindruck über das verwendete Design gewonnen werden. Auf der Anordnung der einzelnen Steuerelemente liegt noch kein großes Augenmerk. Es ist allerdings gut zu erkennen, dass mittels Wiederholung von Farbe und Form eine gewisse Konsistenz im Design entsteht. Die Elemente des Hauptmenüs werden im Untermenü wieder aufgegriffen. Der Benutzer kann somit intuitiv erahnen, wo er im Unterfenster nach Funktionen suchen muss. Da das Farbmenü mit seiner aufrufenden Stelle, der Schaltfläche am linken Rand des Hauptmenüs, farblich wie auch räumlich in Verbindung bleibt, ist die Aufrufstruktur der graphischen Oberfläche besser für den Anwender nachzuvollziehen. Ihm wird durch die Anordnung und die farbliche Gleichheit von Untermenütitelleiste und Menüpunkt am linken Rand stets gezeigt, wie dieses Menü aufgerufen wird und wo es seinen Ursprung hat.

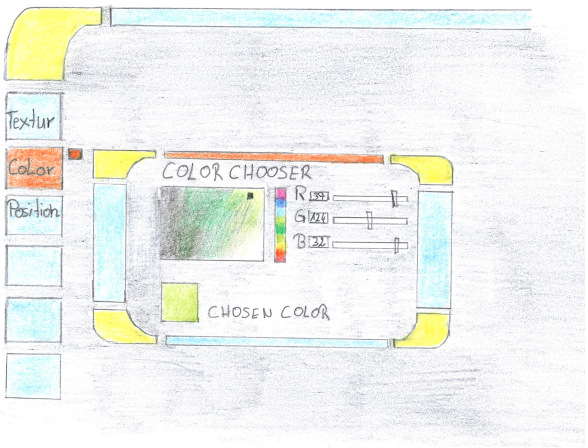


Abbildung 15: Erste Handskizze

Dadurch, dass das Menü an mehreren Stellen unterbrochen ist und nicht komplett bis zum

Bildschirmrand reicht, wirkt es wie ein über die Ausgabe des Shaders gelegter Rahmen. Dieser Eindruck ist beabsichtigt und wird zusätzlich durch die Farbwahl verstärkt. Farben auf schwarzem Untergrund wirken, als ob sie auf diesem „schwebten“. Dies wird auch als *kinetischer Effekt* bezeichnet. [SM00]

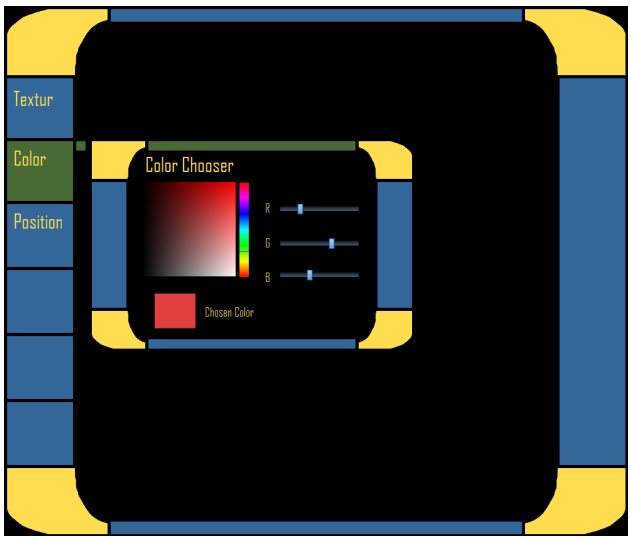


Abbildung 16: Prototyp erstellt mit *Pencil*

In Abschnitt 2.2.4 wurde bereits angesprochen, dass für spätere Skizzen das Prototyping auch mit speziellen Programmen durchgeführt werden kann. In Abbildung 16 ist eine Skizze zu sehen, die mit dem Programm „Pencil“ der Firma „Evolus“ erstellt wurde. Diese Software kann als *Add-In* (Erweiterung) des *Firefox Browsers* oder auch als *Stand-Alone* d.h. eigenständige Software verwendet werden. Pencil ist frei im Internet verfügbar[16]. Beim Erstellen dieser Skizzen zeigte sich, dass Prototyping Tools für graphische Oberflächen, die in GUI Systeme eingebunden sind, effizienter und schneller verwendet werden können als Prototypen für graphische Oberflächen, die ein eigenes Design verfolgen. Zur Realisierung wird hier ein wesentlich höherer Zeitaufwand benötigt.
Das Erstellen von Prototypen ist ein fortlaufender Prozess, in dem Entwürfe immer wieder verbessert werden aber auch komplett verworfen werden können. Vor allem müssen diese Prototypen laufend auf ihre Usability getestet und mit den Vorstellungen des Users abgeglichen werden. Hierfür geeignete Testverfahren wurden in Abschnitt 2.2.5 vorgestellt.
Prototypen werden auch in späteren Phasen der Entwicklung benötigt. Für die einzelnen Menüs und Fenster der graphischen Oberfläche wurden weitere Skizzen erstellt, die jedoch erst nach Ausarbeitung der Struktur der graphischen Oberfläche möglich waren. Im nächsten Abschnitt wird die Farbe der Elemente festgelegt, die in den bisherigen Skizzen noch relativ willkürlich gewählt worden war.

## 4.3 Verwendung von Farben

Die richtige Farbe für eine graphische Oberfläche zu finden ist schwierig, da hierbei sehr stark individuelle Empfinden des Benutzers herein spielen. Auch unterschiedliche kulturelle Hintergründe können einen nicht unerheblichen Einfluss haben.

[16] `http://www.evolus.vn/Pencil/`, zuletzt besucht am 16.03.2009

In der entworfenen graphischen Oberfläche werden sowohl helle Schrift auf dunklem Grund, als auch dunkle Schrift auf hellem Grund eingesetzt. Für bessere Lesbarkeit ist die zweite Variante der ersten vorzuziehen. Wird helle Schrift auf dunklem Grund verwendet, kann der Eindruck beim Benutzer entstehen, dass der Hintergrund in die Schrift hineinfließt und die Farben dabei verschwimmen. Um einen stimmigeren Gesamteindruck und ein besseres Design zu erreichen, wird in diesem Entwurf einer graphischen Oberfläche trotzdem auch helle Schrift auf dunklem Hintergrund verwendet. Dabei wurde darauf geachtet, dass Text- und Hintergrundfarbe einen ausreichenden Kontrast aufweisen und die verwendeten Farben nicht „kollidieren“ und dadurch die Texte für den Benutzer nur schwer lesbar werden. [Joh07]
Im Designprozess wurde auch darauf Rücksicht genommen, dass ein gewisser Anteil von Benutzern genetisch bedingt mehr oder minder stark farbenblind ist. Um die graphische Oberfläche solchen Handicaps gerecht werden zu lassen, wurde eine Oberfläche kreiert, die auch ohne die Verwendung von Farben lesbar ist und bedient werden kann. Farben sind nur ergänzenden Hilfsmittel zur Steigerung der Usability und keine ausschließlichen Bedienungskriterien. Es wird zwar nicht auf Farbeffekte verzichtet, allerdings sind die durch Farben hervorgehobenen Teile beispielsweise zusätzlich durch fettgedruckte Schrift oder andere Effekte kenntlich gemacht. Wie die gewählten Farben auf die Bedienbarkeit von Benutzern mit beeinträchtigter Farbwahrnehmung geprüft werden, ist im Abschnitt 5.4 näher erläutert. [Joh00]
Da die Ausgabe des Shader Viewers meist selbst schon sehr farbenreich ist und im Mittelpunkt des Programms steht, wurde bei der farblichen Gestaltung der Elemente der graphischen Oberfläche eine zurückhaltende Farbgebung angestrebt. Allerdings sollte dies nicht auf Kosten von Lesbarkeit und Usability geschehen. Es wurde zusätzlich versucht, sich auf möglichst wenige unterschiedliche Farben zu beschränken, um den Benutzer nicht zu verwirren und von den graphischen Effekten des Shader Viewers abzulenken. In der Literatur werden maximal fünf bis sieben unterschiedliche Farben empfohlen, die entwickelte graphische GUI kommt mit drei Grundfarben aus. [Nie93]
Des Weiteren ist zu beachten, dass Farben spezielle Eigenschaften symbolisieren können. So wird beispielsweise die Farbe rot mit einer Warnung oder Gefahr in Verbindung gebracht. Diese Assoziation wird in der graphischen Oberfläche genutzt, um kritische Meldungen an den Benutzer zu markieren.
Es gibt viele Regeln zur Farbästhetik, die beschreiben, wie die Wahl von möglichst gut harmonierenden Farben durchgeführt werden soll. Da jedoch das persönliche Empfinden jedes einzelnen Benutzers variiert, wurde bei der Umsetzung der graphischen Oberfläche mit eingeplant, dass die Farbe der graphischen Oberfläche vom Benutzer selbst einstellbar sein soll. Hierdurch wird eine für den Anwender individualisierbare graphische Oberfläche erzeugt. Der Benutzer soll alle in der graphischen Oberfläche verwendeten Farben seinen Ansprüchen entsprechend anpassen können. Allein die rote Farbe zur Kennzeichnung von kritischen Vorgängen

bleibt fest im System gespeichert. Zusätzlich besitzt die graphische Oberfläche einige vordefinierte Vorschläge für die Farbgebung, die dem Anwender zur Auswahl stehen. Dadurch ist der Benutzer nicht gezwungen, jede Farbe einzeln anzupassen, wenn die aktuelle Farbwahl als nicht ansprechend empfunden wird. Er kann ohne großen Zeitaufwand aus einigen Alternativen wählen. Wichtig ist auch, dass dem Anwender die Möglichkeit geboten wird, seine individuelle Farbauswahl zu speichern. Würde er beim Neustart die graphische Oberfläche jedes Mal neu anpassen müssen, wäre dies frustrierend und würde das Konzept der besseren Usability durch eine vom Benutzer individuell anpassbaren Oberfläche zunichte machen.
Zur Entstehung und Auswahl der Farbgebung sind weitere Informationen im Abschnitt 5.4 enthalten. Ein weiterreichendes Konzept der Individualisierung der graphischen Oberfläche durch Skins[17] wird in Abschnitt 4.6.1 vorgestellt.

## 4.4 Struktur der graphischen Oberfläche

Um Elemente einer graphischen Oberfläche zu strukturieren, gibt es mehrere Möglichkeiten. Benutzer empfinden Elemente als zusammengehörend, wenn sie räumlich dicht zusammenliegen, von Linien oder Kästen umrahmt werden, sich gemeinsam bewegen und verändern oder sich aufgrund von Farbe, Form, Größe oder Typographie gleichen. [RP90]
Durch die klare räumliche Trennung der vier Menüleisten des Hauptmenüs wird eine thematische Gruppierung erreicht. Die Menüs verteilen sich dabei auf die vier Ränder der graphischen Oberfläche. Dadurch soll die Orientierung des Benutzers innerhalb der graphischen Oberfläche gefördert werden. Die einzelnen Menüs und Untermenüs sind in sich weiter untergliedert. In Abbildung 17 sind diese Menüleisten durch vier rote Linien voneinander getrennt.
Im Folgenden werden die thematischen Unterschiede der vier Regionen kurz vorgestellt.

**Menü Norden:** Alle Menüs, die die Einstellungen der graphischen Oberfläche betreffen, sind hier untergebracht. Dazu gehören die Farbauswahl der graphischen Oberfläche, die Bildschirmauflösung, die Wahl des verwendeten Skin und Dialoge zum Speichern und Laden von Parametersets.

**Menü Osten:**
Diese Menüregion ist als Informationsmenü gedacht. Hier werden alle Informationen der aktuell ausgewählten Einstellungen und des Shader visualisiert. Es kann zwischen einer Detail- und einer Standardansicht gewählt werden. Bewegt sich die Maus in die entsprechenden Schaltflächen, fahren die einzelnen Informationstafeln heraus und detailliertere Informationen werden verfügbar.

[17] Paket von Bildern und Einstellungen, die das Aussehen und Verhalten von grafischen Benutzeroberflächen festlegen.

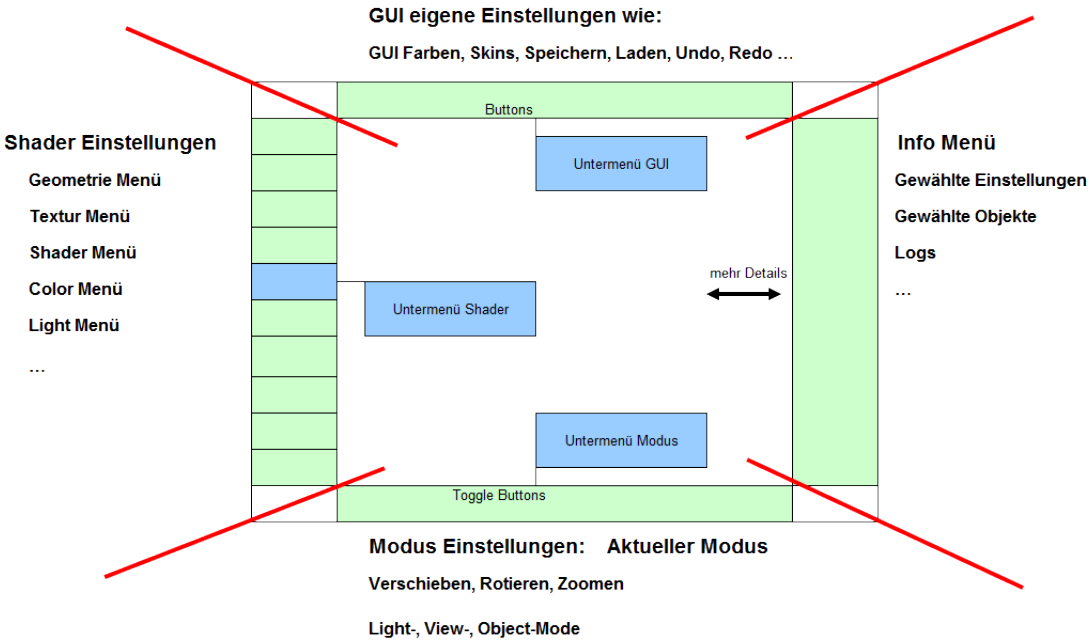


Abbildung 17: Struktur der graphischen Oberfläche

**Menü Süden:**
Dieses Menü informiert den Benutzer darüber, welches Objekt innerhalb der Shader Ausgabe gerade mit der Mause bewegt werden kann. Zusätzlich kann der Anwender aus diesem Menü entnehmen, auf welche Art das gewählt Objekt bewegt werden kann. Beispielsweise könnte sich der Benutzer im Modus befinden, mit dem die Lichtquelle bewegt und eine Rotation hervorgerufen werden kann. Somit erkennt der Anwender sofort, welchen Effekt seine Eingabe auf das dargestellte Objekt im Shader Viewer hat.

**Menü Westen:**
In dieser Menüregion befinden sich alle Parameter und Optionen, die zum dargestellten Objekt und Shader eingestellt werden können. Dabei handelt es sich beispielsweise um Textur oder Farbeinstellungen.

Die Trennung und Gruppierung der graphischen Oberfläche in unterschiedliche Themenbereiche hilft dem Benutzer dabei, sich zu orientieren und zurechtzufinden. Wird eine Funktion gesucht, ahnt er meist schon, in welchem Bereich des Hauptmenüs diese wahrscheinlich zu finden ist. Die genaue Umsetzung der einzelnen Menügruppen wird in Abschnitt 4.5 im Detail betrachtet.
Auch anhand der Untermenüs kann direkt erkannt werden, in welcher thematischen Menügruppe sich der Benutzer gerade befindet. Dies geschieht sowohl durch farbliche Kennzeichnung und eine räumliche Verbindung. Wird ein Untermenü aufgerufen, wird es räumlich an die auf-

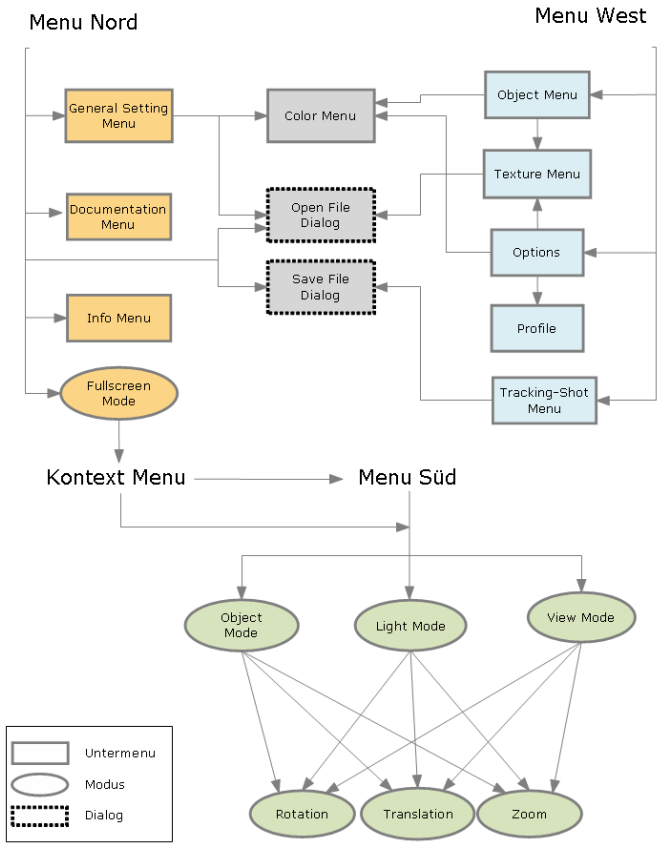


Abbildung 18: Aufrufstruktur der Menüs, Dialoge und Modi

rufende Stelle gebunden. Zusätzlich werden beide farblich markiert.
Abbildung 18 zeigt die Menüführung und die komplette Aufrufstruktur innerhalb der graphischen Oberfläche. Die Hauptmenüregionen, Untermenüs, Dateidialoge und die unterschiedlichen Bewegungsmodi sind dargestellt. Die Regionen Nord, West und Süd des Hauptmenüs sind aufgelistet und es wird dargestellt, welche Untermenüs von dort aufgerufen werden können. Das Hauptmenü Osten wurde nicht mit in die Graphik aufgenommen, da es lediglich Informationen wiedergibt und keine Untermenüs von hier aus angesteuert werden können. Zusätzlich ist allerdings das Kontextmenü zu sehen, das mit einem Klick der rechten Maustaste auf die Ausgabe des Shaders aufgerufen werden kann. Das Kontextmenü spielt vor allem im Vollbildmodus eine entscheidende Rolle, da die restlichen Menüs hier ausgeblendet sind. Aus dem Vollbildmodus heraus soll über das Kontextmenü die Hauptmenüleiste im Süden aufrufbar sein, da diese die Auswahl der unterschiedlichen Bewegungsmodi zur Verfügung stellt, die vor allem auch im Vollbildmodus benötigt werden. Zusätzlich enthält das Kontextmenü Einstellungen zur Bildschirmauflösung. Hier können auch direkt Änderungen im Bewegungsmodus vorgenommen werden.

In graphischen Oberflächen sind redundante Aufrufstrukturen vorteilhaft, wenn sie an geeigneten Stellen eingesetzt werden. Wichtige Menüs oder Optionen sollten nach Möglichkeit immer sofort vom Anwender gefunden und verwendet werden können. Dazu können diese von unterschiedlichen Stellen der graphischen Oberfläche aus aufgerufen werden. Dies ist beispielsweise bei den Bewegungsmodi realisiert. Sie lassen sich sowohl aus dem Kontextmenü, als auch aus der Hauptmenüleiste im Süden auswählen und steuern. Von Einstellungen, die lediglich im Kontextmenü vorgenommen werden können, ist abzuraten. Dieses Menü ist nicht ständig sichtbar und es ist nicht sichergestellt, das der Benutzer weiß, dass ein solches Menü vorhanden ist. Deshalb sind alle im Kontextmenü wählbaren Aktionen auch von anderer Stelle aus in der graphischen Oberfläche erreichbar.
Redundante Aufrufstrukturen spielen bei größeren Programmen mit komplexeren graphischen Oberflächen eine noch entscheidendere Rolle.
Der in Abbildung 18 dargestellte Überblick über die Aufrufstruktur in der graphischen Oberfläche ist auch für die spätere Usabilty der Software von Bedeutung. Der Benutzer muss sich in der Menüstruktur zurecht finden können und nicht lange danach suchen müssen, an welcher Stelle er welche Einstellungen vornehmen kann. Weiterhin ist aus Abbildung 18 zu entnehmen, dass Elemente der graphischen Oberfläche wie das Color Menü in unterschiedlichen Anwendungssituationen wieder aufgerufen werden können. Damit lässt sich der Entwicklungsaufwand reduzieren. Ob die Farbe eines Objektes oder des Hintergrundes geändert werden soll, spielt bei der Darstellung des Menüs zur Farbwahl keine Rolle. Wie aus der Programmstruktur zu ersehen ist, kann auch in einen Vollbildmodus gewechselt werden. Dies ist für ein Programm wie dem Shader Viewer besonders sinnvoll, da die Ausgabe des gerenderten Bildes im Vordergrund stehen soll. Die ausgegebenen Farben sollen nach Möglichkeit ohne störende Einflüsse betrachtet werden können. Große farbige Elemente der graphischen Oberfläche können die vom Shader erzeugten Effekte behindern.
Innerhalb des Vollbildmodus wird lediglich in der oberen rechten Ecke eine kleine Schaltfläche zum Verlassen des Vollbilds angezeigt. Die bei Bedarf über das Kontextmenü zuschaltbare südliche Menüleiste des Hauptmenüs ist dezent gehalten und stört die Ansicht des Shaders nur geringfügig.

## 4.5 Gestaltung der Menüs

Nachdem im letzten Abschnitt die Aufrufstruktur ausgearbeitet wurde, kann der Entwurf der einzelnen Menüs beginnen. Das Design, das bereits in den ersten Skizzen in Abschnitt 4.2 entwickelt wurde, wird nun mit der gewählten Farbe aus Abschnitt 4.3 auf die Menüs angewendet. Es gilt zu überlegen, welche Steuerelemente wie zum Einsatz kommen, damit die Funktionalität bestmöglich unterstützt werden kann und der Benutzer die Menüs möglichst

Abbildung 19: Hauptmenü im Norden

einfach bedienen kann. Die gewählten Elemente müssen an geeigneter Stelle in den Menüs platziert werden. Dabei sollte darauf geachtet werden, dass ein gewisses Maß an Konsistenz eingehalten wird. Vor allem bei einer Software wie dem Shader Viewer, der viele unterschiedliche Parameter verarbeitet, sollten für alle Untermenüs geeignete Konzepte und Strukturen entwickelt werden. Zunächst wird die Umsetzung des Hauptmenüs näher betrachtet und danach sollen exemplarisch zwei ausgewählte Untermenüs vorgestellt werden.

**Hauptmenü Nord:**

In Abbildung 19 ist das Design der Hauptmenüleiste im „Norden“ der graphischen Oberfläche dargestellt. Es wird mit Icons gearbeitet, deren Funktion sich dem Anwender meist intuitiv aufgrund der dargestellten Bilder erschließen. Nach Möglichkeit wurden Metaphern verwendet, die auch aus anderen Programmen bekannt sind, wie beispielsweise die Diskette als Speichersymbol oder die entsprechende Landesflagge zur Auswahl von Sprachen. Sollte sich die Funktion dem Benutzer trotz der Bilder nicht erschließen, stehen Tooltip-Texte für jedes der Symbole zur Verfügung.
Thematisch verwandte oder zusammengehörende Icons wurden durch räumliche Nähe innerhalb der Leiste gruppiert.

**Hauptmenü Ost:**

Die Hauptmenüleiste im „Osten“ teilt sich in mehrere Bereiche, die durch Rahmen voneinander separiert sind. Wie in Abbildung 20 zu sehen ist, besitzt jedes dieser umrahmten Bereiche eine schmale Schaltfläche am Rand, mit der detaillierte Informationen für den Benutzer sichtbar gemacht werden können. Wird auf diese Schaltfläche geklickt, fährt das Menü in den Bildschirm und bleibt dort verankert sichtbar. Durch erneutes Anklicken kann es wieder in den Hintergrund verschoben werden. Dieses Info-Menü wird auch angezeigt, wenn die Maus lediglich innerhalb der Schaltfläche positioniert wird. Allerdings schließt sich das Menü dann auch gleich wieder, sobald der Mauszeiger es wieder verlässt.

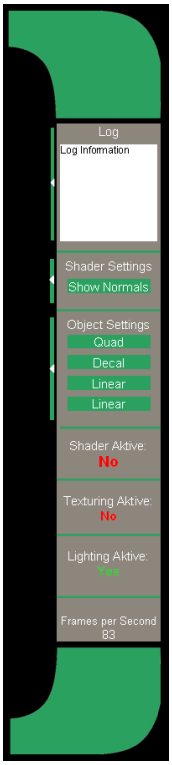


Abbildung 20: Hauptmenü im Osten

Im unteren Teil des östlichen Hauptmenüs wird die aktuell erreichte Frame Rate angezeigt. Diese informiert den Benutzer über die Auslastung des Viewers und wird deshalb durchgehend angezeigt.
Zum Erzeugen schlichter und übersichtlicher Menüs werden Informationen zunächst in Unter-

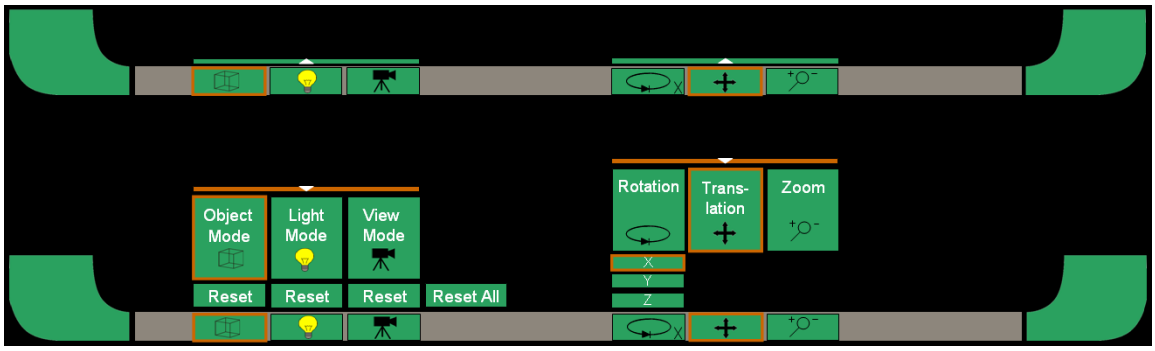


Abbildung 21: Hauptmenü im Süden

menüs „versteckt“. Der Anwender hat die Möglichkeit, sich diese anzeigen zu lassen und vor allem durch Verankerung der Menüs auch permanent zu visualisieren. Die graphische Oberfläche kann gemäß den individuellen Ansprüchen des Benutzers angepasst werden.

**Hauptmenü Süd:**

In diesem Menü werden die Bewegungsmodi verwaltet. Die Auswahlelemente werden am unteren Rand der graphischen Oberfläche platziert. Da sich der Benutzer beim Betrachten des Bildschirms normalerweise eher auf die obere Hälfte konzentriert, sollten hier die wichtigsten Schaltflächen positioniert werden. Die Auswahl der Bewegungsmodi ist für den Anwender zwar wichtig, sie muss jedoch nicht ständig an prominenter Stelle angezeigt werden, da sie leicht durch Tastenkürzel erreicht und zusätzlich durch die sich verändernde Darstellung des Mauszeigers angezeigt werden kann.

Wie in Abbildung 21 zu sehen, lassen sich die Schaltflächen im Süden auf die gleiche Weise bedienen, wie schon für das Hauptmenü im Osten vorgestellt wurde. Werden die Schaltflächen erweitert, können zusätzliche Funktionen ausgewählt werden, wie beispielsweise das Zurücksetzen der einzelnen Bewegungen auf den Ursprungswert. Auch bei diesem Menü wird versucht, ein möglichst schlichtes Design zu erzeugen. Dies ist besonders wichtig, da die südliche Hauptmenüleiste bei Bedarf auch im Vollbildmodus eingeblendet werden kann. Trotz der minimalen Darstellung wird dem Benutzer durch Größe und Farbe ständig angezeigt, welcher Modus gerade aktiv ist. Auch hier wurden wieder Metaphern verwendet, die bei Bedarf durch Tooltip-Texte unterstützt werden können.

Die unterschiedlichen Bewegungsmodi lassen sich zusätzlich mit Tastaturkürzeln ansteuern. Dies erlaubt es dem Anwender, im Vollbildmodus auch ohne Anzeige von Kontextmenü oder der südlichen Hauptmenüleiste zwischen den unterschiedlichen Bewegungsmodi zu wechseln. Hierzu werden Mausbewegungen unter anderem in Verbindung mit den Tasten „STRG“, „ALT“, und „SHIFT“ kombiniert. Die Wahl der Tasten ist für Rechts- wie Linkshänder geeignet, da diese auf beiden Seiten der Tastatur vorhanden sind. Der Benutzer kann dadurch mit der Maus die Bewegungen durchführen und mit der anderen Hand bequem die Tastatur bedienen,

ohne dabei auf die andere Seite der Tastatur greifen zu müssen.
Die Bereitstellung der Shortcuts ist vor allem für den fortgeschrittenen Anwender wichtig, da somit eine schnelle Bedienung der Software ermöglicht wird.
Da das Wechseln der Bewegungsmodi auch mit der Tastatur erfolgen kann, wird dies dem Anwender zusätzlich durch eine veränderte Mauszeigeroptik symbolisiert. Somit kann der Benutzer die gerenderte Ausgabe des Shader Viewers im Blick behalten, während er die abgebildeten Objekte bewegt. Vor allem für Anwender, die mit den unterschiedlichen Tastenkombinationen noch nicht so vertraut sind, ist eine zusätzliche Rückmeldung über den gewählten Modus direkt am Bildschirm sehr angenehm. Wie bereits im Abschnitt 2.2 beschrieben, wird der Benutzer dadurch unterstützt, dass er die Folgen seiner Aktionen für das dargestellte Objekt besser abschätzen kann.

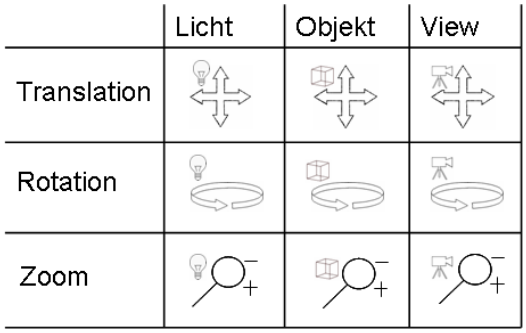


Abbildung 22: Mauszeiger

Eine Auswahl der verfügbaren Mauszeiger ist in Abbildung 22 zu sehen. Anhand der Mauszeigersymbole wird dem Anwender verdeutlich, welches Objekt auf welche Art bewegt werden kann. Man sieht, dass sich die Symbole für Objekt, Lichtquelle und Kamera konsistent in der graphischen Oberfläche wiederholen.

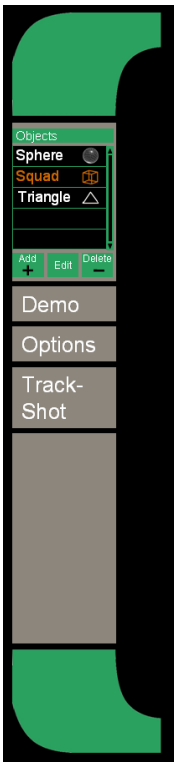


Abbildung 23: Hauptmenü im Westen

**Hauptmenü West**

In Abbildung 23 ist die Hauptmenüleiste im „Westen“ zu sehen. Mit ihrer Hilfe können Einstellungen an den dargestellten Objekten und dem verwendeten Shader vorgenommen werden. Zentrales Element ist die Liste der vorhandenen Objekte, in die Elemente neu aufgenommen oder aus ihr entfernt werden können. Dies geschieht über die mit „+“ und „−“ gekennzeichneten Schaltflächen. Wird ein Element aus der Liste markiert, werden dessen Details in der östlichen Hauptmenüleiste angezeigt. Über die „Edit“ Schaltfläche unterhalb der Liste lässt sich ein Untermenü zum Editieren des Objekts und des darauf angewendeten Shaders aufrufen. Details sind in Untermenüs verborgen, die bei Bedarf aufgerufen werden können.

Bei allen Untermenüs werden konsistent die Schaltflächen zum Abbrechen und Bestätigen eines Vorgangs im unteren rechten Teil des Menüs angeordnet. Wie bei vielen Anwendungen typisch, ist die „OK“ Schaltfläche links von der „Abbrechen“ Schaltfläche angeordnet. Der Benutzer prägt sich die-

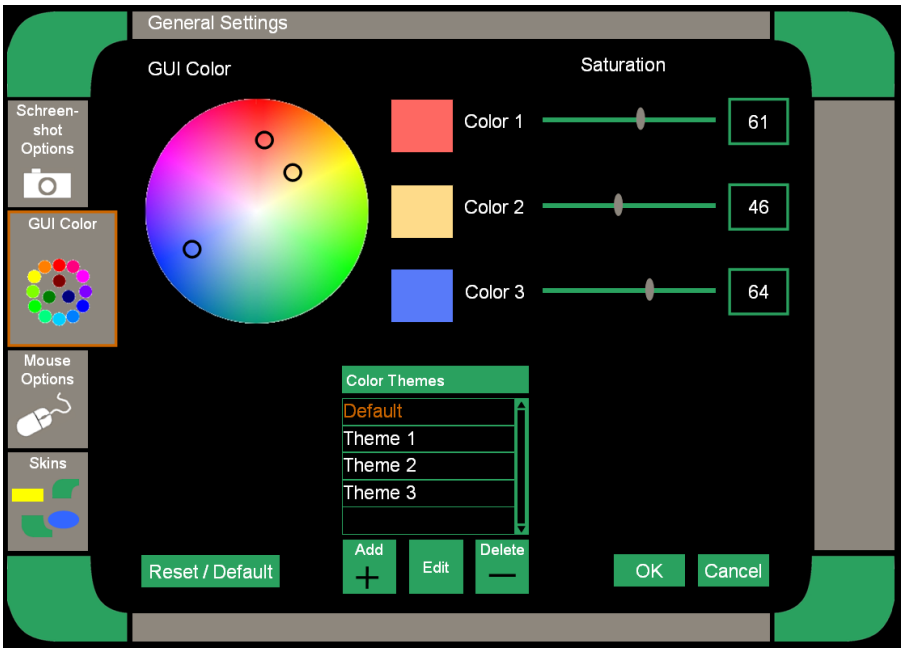


Abbildung 24: General Settings Menu

sen Aufbau schnell ein und findet sich auch schnell in ihm bisher unbekannten Untermenüs zurecht. Die Gefahr von Fehleingaben durch Verwechslung von Tasten wird reduziert.
Können in einem Menü mehrere Inhalte angezeigt werden, ist dies aus der in Abschnitte unterteilten linken Menüleiste ersichtlich, über die zwischen den unterschiedlichen Inhalten gewechselt werden kann. Das gerade angezeigte Element bleibt farblich markiert und wird etwas breiter dargestellt. Wie in Abschnitt 4.3 erläutert, werden zusätzlich zur farblichen Kennzeichnung auch andere Arten der Markierung verwendet, um die graphische Oberfläche besser für Personen mit beeinträchtigter Farbwahrnehmung zugänglich zu machen.

**General Settings Menu**

In Abbildung 24 ist das General Settings Menu dargestellt. Es wird von der Hauptmenüleiste im „Norden“ aufgerufen. Auf der linken Seite sind die unterschiedlichen Inhalte aufgelistet, die im General Settings Menu enthalten sind. Diese Struktur ist analog zum Hauptmenü konzipiert und die Gestaltungsprinzipien setzen sich konsistent in den Untermenüs fort. In der Abbildung ist der Menüpunkt zum Einstellen der GUI Farben ausgewählt. Der Benutzer hat die Möglichkeit, aus einer Liste ein vorgefertigtes Farbschema zu wählen. Wird eines dieser Elemente ausgewählt, wird das Untermenü bereits in diesen Farben dargestellt. Dadurch bekommt der Anwender direkt einen Eindruck, wie sich das ausgewählte Farbschema später auf die Darstellung der graphischen Oberfläche auswirken wird. Zusätzlich kann der Benutzer die drei Farben der graphischen Oberfläche einzeln editieren. Hierzu können die Auswahlkreise innerhalb des dargestellten Farbkreises bewegt werden, um den *Hue* (Farbton) und den *Satu-*

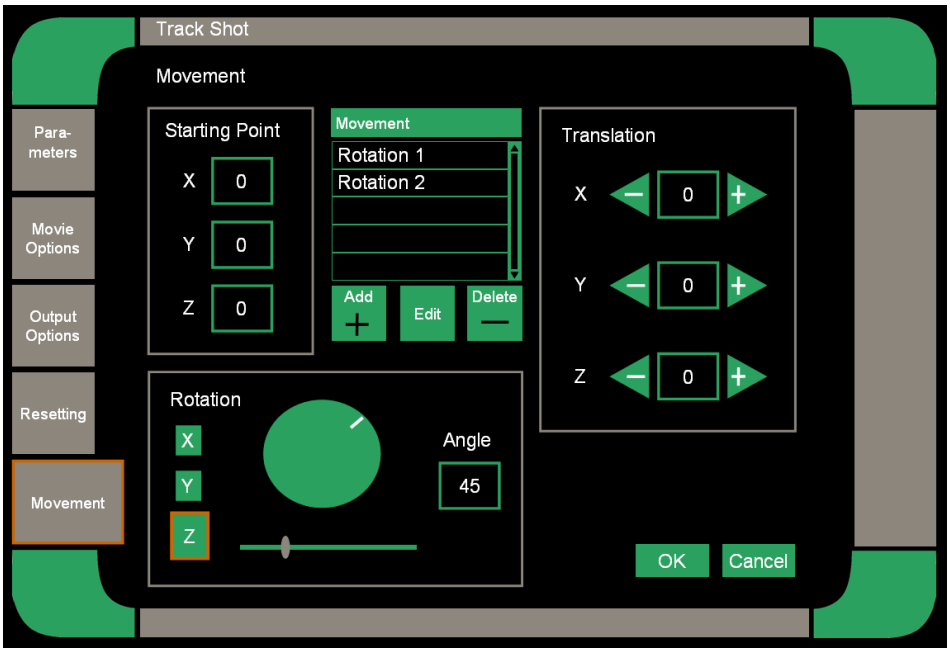


Abbildung 25: Track Shot Menu

*rationswert* (Sättigung) der jeweiligen Farbe einzustellen. Der Wert für *Value* (Hellwert) lässt sich separat mit dem Schieberegler anpassen, der hinter der jeweiligen Farbvorschau verfügbar ist. Mit der „+“, „−“ und „Edit“ Schaltfläche unterhalb der Liste können Farbschemas gespeichert und zur Auswahlliste an verfügbaren Schemen hinzugefügt, gelöscht oder editiert werden.

**Track Shot Menu**

Ein Track Shot stellt eine Art Kamerafahrt innerhalb der gerenderten Szene dar. Mit Hilfe des in Abbildung 25 dargestellten Menüs können Bewegungen im Raum festgelegt werden. Diese werden daraufhin von der Kamera durchgeführt. Wird der Track Shot gestartet, wird eine Bildsequenz erstellt, die während der vorher festgelegten Bewegung innerhalb des Raums entsteht. Es müssen eine Reihe unterschiedlicher Einstellungen vorgenommen werden, wie beispielsweise das Speicherformat und die Auflösung der entstehenden Bilder, um die erstellte Sequenz an die Vorstellungen des Benutzers anzupassen. In Abbildung 25 sind die zur Verfügung stehenden Optionen im linken Rand des Menüs gegliedert dargestellt. Dem Anwender wird in jeder Menügruppe nur eine kleine Auswahl an Einstellungsmöglichkeiten präsentiert. Somit ist es ihm möglich, die Kamerafahrt Schritt für Schritt durch Abarbeitung der einzelnen Menüs zusammenzusetzen.

In der Abbildung ist beispielhaft das Menü zu sehen, mit dem die einzelnen Bewegungselemente festgelegt werden können. Die Gesamtbewegung wird in einer Liste gespeichert, aus der verschiedene Elemente wieder herausgelöscht oder neu eingefügt werden können. Die Bewe-

gungen selbst werden im rechten Teil des Menüs festgelegt. Hierbei kann für Rotationen der Winkel direkt eingegeben werden. Alternativ stehen ein Schieberegler oder ein Drehknopf zur Verfügung. Zusätzlich muss die Achse, um die rotiert werden soll, ausgewählt werden.
Zur Translation werden die X-, Y- und Z-Koordinaten direkt eingegeben oder können mit den Schaltflächen rechts und links vom Eingabefeld herauf- oder herabgesetzt werden. Der Startpunkt der Bewegung wird im linken Teil des Menüs separat gespeichert.
Die Berechnung der Kamerafahrt kann relativ viel Zeit in Anspruch nehmen, da viele Bilder erzeugt und gespeichert werden müssen. Dem Benutzer wird über einen Fortschrittsbalken der aktuelle Stand der Berechnung angezeigt. So wird der Anwender informiert, dass die Aufgabe ausgeführt wird und gleichzeitig kann er abschätzen, wie lange er noch auf das Ergebnis warten muss. Da bei Fehleingaben oder Unwissenheit des Benutzers die Bearbeitungszeit wesentlich länger dauern kann, als von ihm erwartet, gibt es eine Möglichkeit, die Berechnung über eine Schaltfläche abzubrechen. Wichtig ist, dass das Menü nach Abbruch in den Ausgangszustand vor der Berechnung der Kamerafahrt zurückgesetzt wird.

Die graphische Oberfläche besteht aus zahlreichen weiteren Menüs, die hier allerdings nicht alle vorgestellt werden können. Struktur, Aufbau und Design werden auch hier konsistent fortgesetzt. Bei den in diesem Abschnitt dargestellten Abbildungen handelt es sich um das Standardskin der graphischen Oberfläche. Das Prinzip von Skins wird in Abschnitt 4.6.1 näher erläutert.

## 4.6 Weitere Features

Im Folgenden werden einige weitere Eigenschaften und Konzepte der graphischen Oberfläche angesprochen und erläutert.

### 4.6.1 Skins

Da die Gestaltung graphischer Oberflächen unterschiedlich ansprechend auf Benutzer wirkt, ist es von Vorteil, dem Anwender gewisse Gestaltungsfreiheiten zu bieten. Damit kann er die Oberfläche seinen eigenen Anforderungen und Vorstellungen entsprechend gestalten und individualisieren. Ein erster Ansatz wurde bereits in Abschnitt 4.3 besprochen. Das Ändern der Farben der graphischen Oberfläche ist eine solche Möglichkeit. Findet jedoch die Anordnung und Gestaltung der einzelnen Menüs keinen Anklang, kann mit der Anpassung von Farben kein großer Mehrwert für den Benutzer erzielt werden.
Um auch letzteren Wünschen zumindest teilweise gerecht zu werden, wurden bei der Entwicklung der graphischen Oberfläche des Shader Viewers zwei alternativ auswählbare Skins eingeplant. Dies ermöglicht eine alternative optische Darstellung, die Funktionalität bleibt

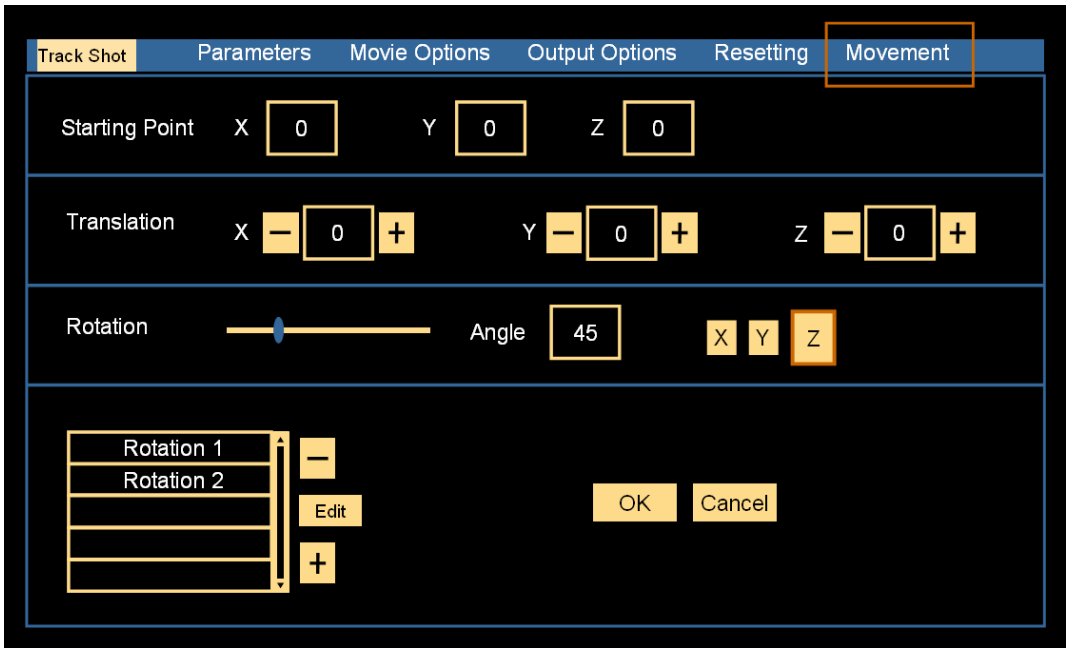


Abbildung 26: Screenshot Skin

erhalten. Skins können sich in Art und Umfang ihrer Veränderungen stark unterscheiden. Einige nehmen nur kleinere Veränderungen vor, um die graphische Oberfläche ästhetischer zu gestalten. Andere hingegen verändern das komplette Look and Feel der Software. So können Elemente beispielsweise mit dem Ziel neu positioniert werden, die Software dadurch besser bedienbar zu machen.

In der hier präsentierten graphischen Oberfläche wurde nur eine alternative Skin mit einigen optischen Änderungen implementiert. Ist eine Software allerdings erst einmal „skinnable“ gemacht, d.h. mit verschiedenen Skins darstellbar, dann ist es wesentlich einfacher im Nachhinein weitere optische Erscheinungsbilder für diese Software zu entwickeln. Die nötigen Schnittstellen sind bereits angelegt und eine spätere Ergänzung durch weitere Erscheinungsformen wird vereinfacht. Dies ist bei diesem Entwurf auch der Grundgedanke für das Einbinden eines weiteren Skins.

Befindet sich die Software längere Zeit im Einsatz, können sich Schwächen in der graphischen Oberfläche zeigen, die bei frühen Tests nicht aufgefallen sind. Handelt es sich dabei um ungünstige Platzierungen von Steuerelementen oder Menüstrukturen, dann kann eventuell die Entwicklung einer neuen Skin Abhilfe schaffen.

Ein Ausschnitt der alternativ verfügbaren Skin ist in Abbildung 26 zu sehen.

### 4.6.2 Parametersets

Da während der Programmanwendung eine Reihe von benutzerspezifischen Einstellungen vorgenommen werden, bietet es sich an, diese zu speichern. Dadurch können bei weiteren Anwendungen der Software die neuen Einstellungen direkt geladen werden. Dies erlaubt es dem Benutzer, unterbrochene Arbeiten an genau derselben Stelle zu einem späteren Zeitpunkt fortzusetzen.
Prinzipiell lassen sich zwei verschiedene Arten von Daten unterscheiden. Einerseits gibt es die shader- und objektspezifischen Daten wie beispielsweise die verwendeten Texturen, die Position der Lichtquelle oder das ausgewählte Objekt. Auf der anderen Seite gibt es auch Einstellungen und Parameter, die sich auf die graphische Oberfläche beziehen. Hierzu zählen beispielsweise die Bildschirmauflösung, die Wahl der Hintergrundfarbe oder das ausgewählte Skin.
Deshalb besteht die Möglichkeit, diese Daten separat zu speichern. Beim Speichern der Daten kann der Benutzer dann wählen, ob er nur die Shader- und Objektdaten oder auch die Einstellungen, die die graphische Oberfläche betreffen, speichern möchte. Das Laden funktioniert analog.

### 4.6.3 Unterstützung mehrerer Sprachen

Um die Software für eine größere Anzahl von Benutzern zugänglich zu machen, kann die graphische Oberfläche in englischer und deutscher Sprache dargestellt werden. Viele Begriffe im Bereich der Shaderentwicklung sind ohnehin dem Angelsächsischen entlehnt und daher ist die bloße Übersetzung ins Englische mit wenig Aufwand verbunden. Für unerfahrene Nutzer der Software, mit nur geringen Englischkenntnissen ist die Option einer deutschen Menüführung jedoch eine große Erleichterung. Vor allem in Bezug auf Hilfe und Dokumentation kann dies sehr hilfreich sein.

# 5 Umsetzung

Dieses Kapitel befasst sich mit der Umsetzung der ausgearbeiteten Konzepte. Es soll aufgezeigt werden, welche Probleme bei der Realisierung aufgetreten sind und wie diese gelöst werden konnten. Verschiedene Alternativen in der Programmierung werden erörtert und ihre Vor- und Nachteile gegeneinander abgewogen.

## 5.1 Wahl der Programmiersprache / Schnittstelle

Da die entwickelte graphische Oberfläche und die Ausgabe des Shader Viewers später als eine Einheit wirken sollen, war eine Analyse der Programmierung des Shader Viewers notwendig. Dadurch können spätere Probleme bei der Verbindung und Integration der beiden Programmteile vermieden werden. Die vom Shader Viewer bereitgestellten Schnittstellen müssen genau bekannt sein, um die zu entwickelnde graphischen Oberfläche darauf auszurichten und anzupassen. Alle Funktionen des Shader Viewers sollten von der graphischen Oberfläche aus realisierbar sein und unterstützt werden. Probleme könnten beispielsweise bei der Berechnung auftreten, wenn unterschiedliche Einheiten oder Datentypen verwendet wurden. Eventuell müssen Umrechnungen vorgenommen werden, bei denen auch auf Rundungsprobleme geachtet werden muss.
Der Shader Viewer wurde in *C++*[18] und *OpenGL* implementiert. Auf OpenGL wird in Abschnitt 5.1.1 näher eingegangen.
Die ursprüngliche graphische Oberfläche des Shader Viewers besteht aus *.NET* Standardkomponenten, die eine schnelle Entwicklung einer graphischen Oberfläche möglich machen. Das .NET Framework wird in Abschnitt 5.1.3 vorgestellt.
Im Folgenden wird OpenGL etwas näher erläutert und analysiert, ob es sich zur Erstellung der graphischen Oberfläche eignet und anbietet.

### 5.1.1 OpenGL

„Open Graphics Library“ (OpenGL) ist eine Spezifikation einer Programmierschnittstelle. Sie ist plattform- und programmiersprachenunabhängig und zur Entwicklung zwei- und dreidimensionaler Applikationen gut geeignet. Durch die Unabhängigkeit von System und Sprache ist OpenGL vielfältig einsetzbar und portabel. Im professionellen Bereich wird OpenGL als Standard vielfach eingesetzt.
Es stellt sich die Frage, ob OpenGL allen Anforderungen bei der Entwicklung der graphischen Oberfläche gerecht wird. Im Folgenden sind ein paar Eigenschaften von OpenGL aufgelistet, die für die Realisierung der graphischen Oberfläche auch hier benötigt werden.

[18] Standardisierte höhere Programmiersprache, die Programmierparadigmen wie Objektorientierung, generische- und prozedurale Programmierung unterstützt.

- Zwei- und dreidimensionale Objekte können erzeugt und graphisch dargestellt werden.
- Das Texturieren von Flächen wird unterstützt.
- Die weite Verbreitung garantiert eine gute Dokumentation und ausreichend Support. Leicht beziehbare Beispielprogramme und Hilfestellungen senken den Entwicklungsaufwand.
- OpenGL kann relativ schnell erlernt werden und erfordert geringe Einarbeitungszeit.
- Durch die ständige Weiterentwicklung ist OpenGL auf dem neusten Stand der Technik.
- Unproblematische Verwendung auf aktueller Hardware, vor allem Graphikkarten werden unterstützt.
- Entwicklung von schnellen und effizienten Lösungen möglich.
- Hohe visuelle Qualität der dargestellten Elemente ist gewährleistet.
- Stabilität und Robustheit sind gegeben.
- Abwärtskompatibilität ist vorhanden, um sicherzustellen, dass die entwickelte Software auch von neueren OpenGL Versionen unterstützt wird.

[WNDS99]

Da alle geforderten Eigenschaften zur Umsetzung der graphischen Oberfläche erfüllt und die Kompatibilität zum Shader Viewer sichergestellt ist, kann OpenGL zur Implementierung verwendet werden.

### 5.1.2 GLUT

Da eine graphische Oberfläche aus mehreren Menüs besteht und die Eingaben des Benutzers via Maus oder Tastatur weiterverarbeitet werden müssen, erschien das „OpenGL Utility Toolkit“ (GLUT) zunächst als eine interessante Ergänzung. Mithilfe dieses *Toolkits*[19] lassen sich schnell und einfach Fenster erzeugen, die später als Menüs verwendet werden können. Tastatur- und Mauseingaben können für jedes der erzeugten Fenster einzeln abgefragt und der Umgang (Eventhandling) mit diesen somit erleichtert werden. Vorgefertigte Funktionen erleichtern es, Ereignisse (Events) von Tastatur und Maus zu differenzieren. Beispielsweise kann ermittelt werden, welche Taste der Maus ein Event ausgelöst hat, und ob dieses beim Drücken oder beim Loslassen der Taste generiert wurde. Es stehen auch Funktionen zur Verfügung, um Mausbewegungen (Mouse Motion) genauer zu ermitteln. Diese Eigenschaften werden für die Umsetzung der graphischen Oberfläche benötigt, da die Menüs unterschiedlich auf die entsprechenden Eingaben reagieren. Dies ist beispielsweise am Kontextmenü zu sehen, das lediglich aufgerufen wird, wenn die rechte Maustaste gedrückt wird. Beim Drücken der linken

[19] Sammlung von Bibliotheken, Klassen und Schnittstellen, die das Erstellen von Computerprogrammen vereinfacht

Maustaste erscheint das Menü jedoch nicht.
Zusätzlich unterstützt das GLUT unter anderem Textausgaben auf dem Bildschirm, das Erstellen einfacher Kontextmenüs und unterschiedliche Mauszeiger.
Auf den ersten Blick schien dies genau das richtige Paket, um die Standard OpenGL Funktionen zu erweitern. Der erste Ansatz der entwickelten graphischen Oberfläche wurde auch unter Verwendung von GLUT Funktionen erstellt.
Es stellte sich dann aber schnell heraus, dass das GLUT zu viele Nachteile aufweist und nicht ausgereift genug ist, um in der graphischen Oberfläche eingesetzt zu werden. Die neuste Version des GLUT für Windows datiert noch aus dem Jahr 2001 und wird aktuell nicht mehr weiterentwickelt. Daher sind diesbezüglich auch keine Verbesserungen in nächster Zeit zu erwarten.
Das GLUT ist nicht für größere Projekte geeignet und kein *Full-Featured Toolkit*. Dies bedeutet, dass einige Funktionen und Eigenschaften nicht unterstützt werden, die jedoch eigentlich zum Umfang eines Toolkits gehören, das die Erstellung von Unterfenstern ermöglicht. Ein entscheidender Nachteil des GLUT ist auch, dass es nicht auf Performance, also Geschwindigkeit, in der Ausführung optimiert wurde. Das Laufzeitverhalten ist stark systemabhängig und nur schwer vorhersagbar.
Es gibt zwar zahlreiche weitere Alternativen zum GLUT, jedoch konnte keine vollständig überzeugen. Bei der weiteren Entwicklung wurde deshalb auf Toolkits, die die Programmierung und Handhabung von Fenstern unterstützen, verzichtet. [WNDS99], [Kil96]

### 5.1.3 .NET Framework

.NET ist eine Softwareentwicklungsplattform der Firma Microsoft. Sie umfasst eine Laufzeitumgebung und eine Sammlung von Klassenbibliotheken und Dienstprogrammen. Die vorherige graphische Oberfläche des Shaders, die in Abbildung 12 zu sehen ist, wurde mit „Windows Forms“ kreiert. Diese Programmierschnittstelle zur Erstellung graphischer Oberflächen ist Teil des .NET Frameworks. Sie erlaubt den Zugriff auf Steuerelemente, die das Microsoft Windows Look and Feel unterstützen. Diese können in Entwicklungsumgebungen wie „Microsoft Visual Studio“ zu graphischen Oberflächen zusammengefügt werden. Im Programmcode kann auf die Steuerelemente zugegriffen, ihre aktuellen Werte abgefragt oder auf auftretende Ereignisse reagiert werden. Dem Entwickler wird ein Großteil der Arbeit zur Erstellung der graphischen Oberfläche abgenommen, da Steuerelemente nicht selbst entwickelt und deren grundlegende Funktionen implementiert werden müssen. Die Reaktion auf unterschiedliche Ereignisse, die vom Steuerelement beispielsweise durch Modifikation mit Tastatur oder Pointing Device ausgelöst werden können, ist bereits vordefiniert. Der Programmierer kann sich somit hauptsächlich auf die eigentliche Funktionalität des Programms konzentrieren und muss nicht auch noch auf das optische Design der graphischen Oberfläche und seiner Elemente achten.

Unter Zuhilfenahme von Windows Forms erlaub das .NET Frameworks zwar eine schnelle und unkomplizierte Entwicklung von graphischen Oberflächen, jedoch lässt dies für die Entwicklung neuer Designkonzepte wenig Spielraum. Die Ausgabe des Shader Viewers erscheint durch die Standardkomponenten nicht wirklich in die graphische Oberfläche integriert zu sein. Weitere Vor- und Nachteile sind aus Abschnitt 3.1.3 zu entnehmen. Um die im Kapitel 4 eingeführten Konzepte umzusetzen, ist die Verwendung von Windows Forms ungeeignet und kommt daher bei der Implementierung der graphischen Oberfläche nur bedingt zum Einsatz. Das .NET Framewok in Kombination mit Visual Studio 2008 bietet allerdings eine komfortable Programmierumgebung. Diese unterstützt auch die Programmierung in OpenGL und wurde deshalb zur Umsetzung der aktuellen graphischen Oberfläche gewählt. In .NET lässt sich mit unterschiedlichen Sprachen programmieren wie beispielsweise *Visual Basic .NET*, *C++/CLI*, *C#* oder *J#*. [Sch05]
Im nächsten Abschnitt wird die gewählte Sprache C++/CLI etwas näher beschrieben.

### 5.1.4 Umsetzung der GUI mit C++/CLI und dem .NET Framework

Durch die Wahl von OpenGL und des .NET Frameworks war bis hierhin noch kein Festlegung auf eine Programmiersprache erforderlich.
Der laufzeitkritische Teil der Software ist beim Shader Viewer klar in der Programmierung der Ausgabe des eigentlichen Viewers zu suchen. Die graphische Oberfläche gibt Daten und Befehle an den Shader Viewer weiter, der daraufhin die aufwendigen Berechnungen der Ausgabe ausführt, bis das Bild fertig gerendert am Bildschirm ausgegeben werden kann.
Die Berechnung graphischer Effekte kann sehr zeitaufwendig sein. Können Bewegungen von Objekten innerhalb der gerenderten Ausgabe nicht flüssig dargestellt und ausgeführt werden, oder muss mit längeren Wartezeiten bei der Berechnung von Effekten gerechnet werden, ist die Benutzung der Software für den Anwender nicht zufrieden stellend.
Im Folgenden werden ein kritische Grenzwerte für Antwortzeiten von Computerprogrammen gezeigt und wie diese auf Anwender wirken.
Bei einer Antwortzeit der Software die weniger als 0.1 Sekunden beträgt, hat der Benutzer das Gefühl, die Software läuft flüssig.
Bei Antwortzeiten, die zwischen 0.1 Sekunden und 1.0 Sekunden liegen, merkt der Anwender die Verzögerung. Er spürt aber noch keine Störung in seinen Arbeitsabläufen.
Bei längeren Wartezeiten geht der Arbeitsfluss verloren. Der Benutzer sieht die Software als umständlich und zeitraubend an. [Nie93]
Aus diesem Grund wurde der Shader Viewer in C++ umgesetzt. Mit dieser Programmiersprache können effiziente und schnelle Programme geschrieben werden. Dies liegt unter anderem

daran, dass in die Entwicklung von C++ *Compilern*[20] bereits seit längerer Zeit viel Entwicklungsarbeit geflossen ist und mit C++ eine relativ systemnahe Programmierung möglich ist. Aus Kompatibilitätsgründen und der dadurch einfacheren Umsetzung der Entwicklung wurde auch die graphische Oberfläche in C++ implementiert.
Weiterhin wird im .NET Framework programmiert, da hier die Möglichkeit besteht, den Inhalt eines OpenGL Programms in ein Windows Forms Panel oder ein ähnliches Element zu rendern.
Events von Tastatur oder Pointing Device können weiterhin von dem Windows Forms Steuerelement empfangen werden, auf dem der OpenGL Inhalt gerendert wird und an die graphische Oberfläche weitergeleitet werden. Diese Events werden daraufhin geprüft, zu welchem Steuerelement sie gehören. Ist das Steuerelement ermittelt, muss zusätzlich überprüft werden, ob dieses überhaupt auf das spezifische Event reagieren kann. Ein Schieberegler reagiert beispielsweise auf Mausbewegungen mit gedrückter Maustaste, wohingegen ein Button nur auf Mausklicks anspricht. Bei Bedarf werden die Events zu schnittstellenkonformen Daten weiterverarbeitet, die dann an den Shader Viewer weitergegeben werden können. Wenn ein Schieberegler vom Anwender bewegt wird, der für die Eingabe von Winkeln zuständig ist, müssen zunächst die empfangenen X- und Y-Koordinaten in Werte zwischen 0° und 360° umgerechnet werden, bevor diese weiterverarbeitet werden können. Die aktualisierte Darstellung der graphischen Oberfläche muss natürlich neben den an den Shader Viewer übergebenen Daten berechnet werden.
Da sowohl Shader Viewer, wie auch die graphische Oberfläche mit OpenGL und .NET arbeiten, besteht die Möglichkeit, die beiden Programmteile zusammen in ein Panel der Windows Forms zu rendern.
Vorteilhaft ist auch eine im .NET Framework verfügbare Spracherweiterung von C++. Sie wurde von Microsoft entwickelt und nennt sich *C++/CLI*. CLI steht für *Common Language Infrastructure* und ist ein Standard zur Spezifikation von Systemen.
C++/CLI ist speziell auf .NET zugeschnitten und bringt einige Veränderungen zum Standard C++ mit sich. Durch diese Spracherweiterung werden Teile der Programmierung vereinfacht. Hierunter fällt beispielsweise die automatische Speicherverwaltung, bei der nicht mehr benötigter Speicherplatz automatisch wieder freigegeben wird. Außerdem wurden Aufzählungstypen, so genannte *Enumerations* verbessert und *Delegates* mit in die Sprache aufgenommen. Delegates sind spezielle Variablen, die eine Methode einer Klasse referenzieren. Sie lassen sich besonders gut für die Eventverarbeitung einsetzen.
Die Umsetzung der graphischen Oberfläche erfolgt aufgrund der oben angeführten Aspekte in C++/CLI.

[20] Kompilierer sind Computerprogramme, die ein in Quellsprache geschriebenes Programm in ein Zielprogramm umwandeln, das meist in einer Assemblersprache geschrieben wird.

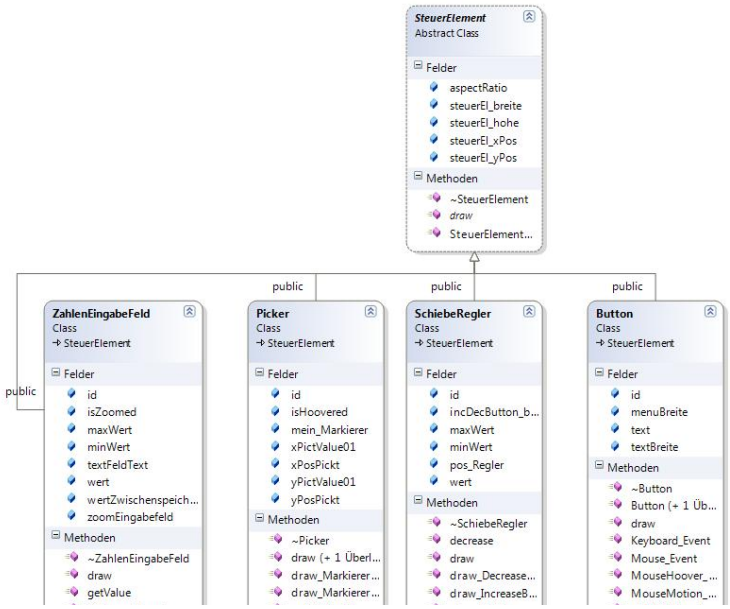


Abbildung 27: Vererbungshierarchie Steuerelemente

## 5.2 Widgets und OOP

Da Funktionalität und Design der benötigten Steuerelemente neu erstellt werden müssen, bietet es sich an, eine geeignete Struktur zu wählen, die den Programmieraufwand begrenzt. Der objektorientierte Ansatz ist hier besonders viel versprechend, da die Steuerelemente in vielerlei Hinsicht ähnliche Funktionen und Eigenschaften aufweisen. Die Verwendeung der Programmiersprache C++ erlaubt die Umsetzung *objektorientierter Programmierung* (OOP).
Zunächst muss ermittelt werden, welche Steuerelemente benötigt werden und wie sich diese in eine Vererbungshierarchie zusammenfügen lassen. Abbildung 27 zeigt einen Ausschnitt aus einer solchen Vererbungshierarchie in einem Klassendiagramm dargestellt. Das Wurzelelement *Widget* (in der Abbildung Steuerelement) vereinigt alle Eigenschaften und Funktionen, die von jedem der Elemente benötigt werden. Dies sind beispielsweise Position, Höhe, Breite und eine *draw* Methode, die das Element zeichnet. Diese Wurzel kann als *abstrakte Klasse* implementiert werden, da keine Objekte vom Typ Widget angelegt werden müssen. Lediglich von den erbenden spezifischen Steuerelementklassen sollen später Instanzen angelegt werden. Verfolgt man die Baumstruktur nach unten, kommen weitere Eigenschaften und Methoden hinzu, die die jeweiligen Steuerelemente charakterisieren. Verschiedene Äste gruppieren gleichartige Steuerelemente, die ähnliche Methoden und Eigenschaften aufweisen.
Beispielsweise bietet es sich an, dass ein Zahleneingabefeld von einem Texteingabefeld abgeleitet wird. Viele Methoden und Attribute können übernommen werden. Zusätzlich müssen für das Zahleneingabefeld zwei Schaltflächen angefügt werden, die die dargestellte Zahl um eine Einheit erhöhen oder vermindern können. Darüber hinaus müssen noch Anpassungen bei den

akzeptierten Zeichen vorgenommen werden.
Durch den objektorientierten Ansatz entsteht eine Art eigenes Framework, dessen Elemente an mehreren Stellen der graphischen Oberfläche wieder verwendet werden können. Elemente, die sich weiter unten im Baum befinden, müssen lediglich ihre neue hinzukommenden Eigenschaften und Methoden implementieren. Werden Methoden auf unterschiedliche Art implementiert, wie beispielsweise *getValue*, die den aktuellen Wert des Steuerelements zurückgibt, kann diese von der jeweiligen Steuerelementklasse überschrieben werden. Somit lässt sich jedes Steuerelement mit getValue aufrufen und der aktuelle Wert des Elements wird zurückgeliefert. Dies ist unabhängig davon, wie der Wert innerhalb des Steuerelements berechnet wird. Ein Schieberegler errechnet beispielsweise aus der Position und der Gesamtlänge, welcher Wert gerade eingestellt ist. Eine Liste hingegen gibt als Wert das aktuell markierte Element zurück.
Dieses Prinzip wird als *Polymorphismus* bezeichnet. Methodenaufrufe können auf Objekten verschiedener Klassen innerhalb der Vererbungshierarchie ausgeführt werden, wobei diese auf den Aufruf unterschiedlich reagieren.
Wie der Wert vom jeweiligen Steuerelement erzeugt wird, bleibt für den Aufrufer verborgen. Der Wert kann durch die vorgegebene Schnittstelle, also die Methode getValue abgerufen werden. Diese Prinzipien nennt man *Datenkapselung*.
Durch den objektorientierten Ansatz lässt sich der Programmieraufwand reduzieren und eine übersichtliche Struktur der verfügbaren Steuerelemente entsteht. Sind die Steuerelementklassen erst einmal angelegt, können beliebig viele Instanzen der jeweiligen Elemente erzeugt werden und in der graphischen Oberfläche zum Einsatz kommen. Instanzen der gleichen Klasse können unterschiedliche Eigenschaften aufweisen, je nachdem welche Attribute ihnen beim Anlegen über ihren *Konstruktor* gegeben wurden. Dies betrifft beispielsweise die Größe, den Wertebereich oder die Beschriftung.
Programmierter Code kann innerhalb der Vererbungshierarchie und durch Anlegen mehrerer Instanzen einer Klasse wieder verwendet werden. Das Framework ist leichter erweiterbar, falls neue bisher noch nicht implementierte Steuerelemente benötigt werden.

## 5.3 Umsetzung ausgewählter Features

In diesem Abschnitt wird die Umsetzung einiger ausgewählter Features der graphischen Oberfläche erläutert.

### 5.3.1 XML-Import von Skins

Zur Realisierung von unterschiedlichen Skins in der graphischen Oberfläche werden die für die Oberfläche relevanten Daten aus *Extensible Markup Language* (XML) Dateien ausgelesen. In diesen Dateien lassen sich Informationen übersichtlich und strukturiert speichern. Einzelne Werte werden innerhalb von vordefinierten *Tags* untergebracht. Dies sind spezielle Beginn-

und Endkennungen, mit deren Hilfe die hierarchische Baumstruktur des XML Dokuments entsteht. Für ein Steuerelement könnte eine solche Struktur beispielsweise folgendermaßen aussehen:

```
<widget>
   <type>button</type>
   <caption>exit</caption>
   <position>
       <x_Pos>0.2</x_Pos>
       <y_Pos>0.1</y_Pos>
   </position>
    ...
</widget>
```

Die entstehende Baumstruktur wird von der graphischen Oberfläche ausgelesen. Dadurch können alle benötigten Eigenschaften wie Positionen, Größen, Texturen, usw. gespeichert werden. Beim Erstellen eines Skins müssen jedoch gewisse Regeln genau eingehalten werden. Informationen müssen zwischen den richtigen Tags an der richtigen Stelle im Dokument untergebracht werden. Zusätzlich ist auch auf Einhaltung der richtigen Einheiten der Daten zu achten.
Durch die Beschriftung der Tags ist das XML Dokument relativ leicht verständlich und an vielen Stellen selbsterklärend. Bei langen Dokumenten können Ebenen der Baumstruktur ausgeblendet werden, um den Überblick zu wahren.
Wegen der übersichtlichen Strukturierung der Daten wurde für die Umsetzung der graphischen Oberfläche dieses Speicherformat für die Skins ausgewählt.

### 5.3.2 Redo und Undo

Dem Benutzer soll es ermöglicht werden, seine zuletzt ausgeführten Aktionen rückgängig zu machen. Dies ist ein wichtiges Usability-Kriterium, das bereits in Abschnitt 2.2.2 angesprochen wurde. Rückgängig gemachte Eingaben sollen auch wiederhergestellt werden können, wenn zwischenzeitlich keine neuen Aktionen stattgefunden haben.

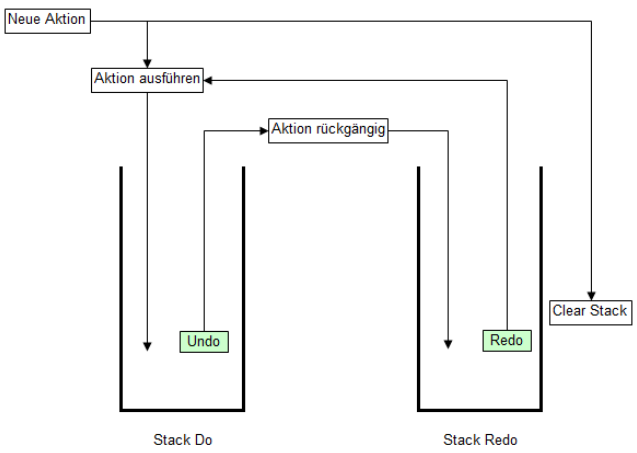


Abbildung 28: Undo / Redo

In Abbildung 28 ist die Speicherstruktur dargestellt, die diese Funktionalität der Software ermöglicht. Es werden zwei *Stacks*[21] angelegt. In dem Linken werden die ausgeführten Aktionen (Stack Do) und in dem Rechten die rückgängig gemachten Aktionen (Stack Redo) gespeichert. Man kann anhand der Abbildung erkennen, dass neu ausgeführte Befehle im „Stack Do“ gespeichert werden. Gleichzeitig wird der „Redo Stack“ gelöscht (Clear Stack). Beim Rückgängig machen einer Aktion (Undo), wird diese in den „Stack Redo“ verschoben. Dieser speichert den Befehl so lange zwischen, bis eine neue Aktion stattfindet, oder das Ergebnis der letzten Arbeitsschritte über den „Redo“ Befehl (Redo) wieder in den „Stack Do“ verschoben wird.

In Abbildung 29 ist ein kleines Beispiel wiedergegeben, welches noch einmal die Funktionsweise verdeutlichen soll. Zunächst werden zwei neue Aktionen A und B ausgeführt und im „Stack Do“ gespeichert. „Undo“ verschiebt den zuletzt ausgeführten Arbeitsschritt B in den „Stack Redo“. Ein weiteres „Undo“ transferiert auch Aktion A in den rechten Stack. „Redo“ verschiebt den zuletzt rückgängig gemachten Arbeitsschritt A wieder in den „Stack Do“. Da im nächsten Schritt eine neue Eingabe C ausgeführt wird, kann der „Stack Redo“ gelöscht und C in den „Stack Do“ gelegt werden. Der „Stack Redo“ wird gelöscht, da ab jetzt keine alte Aktion wiederhergestellt werden darf.

Die Größe der beiden Stacks ist ausschlaggebend dafür, wie viele Arbeitsschritte rückgängig gemacht oder wiederhergestellt werden können. Der „Stack Do“ sollte dabei über eine höhere Kapazität verfügen, da er meist mehr Aktionen speichern

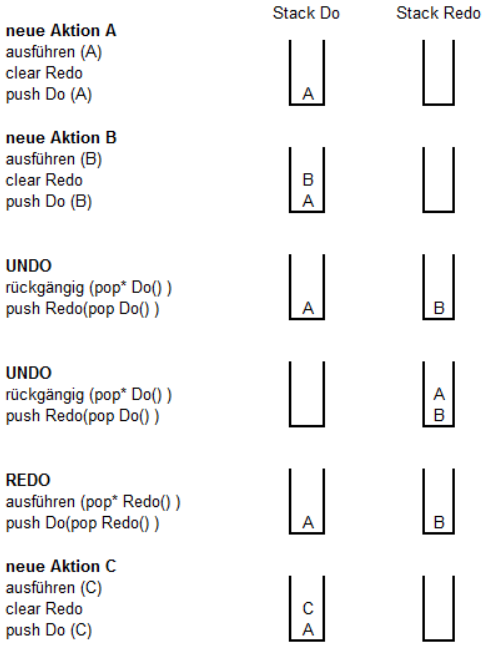


Abbildung 29: Beispiel Undo / Redo

[21] Kellerspeicher, Datenstruktur in der Informatik

muss. Ist ein Stack voll, darf nicht der gewöhnliche *pop* Befehl ausgeführt werden, um Platz für ein neues Element zu schaffen. Der Eintrag, der am weitesten unten im Stack liegt, muss entfernt werden und alle anderen Elemente müssen eine Position nach unten „wandern“. Deshalb wird hier eine Datenstruktur benötigt, die nicht nur die bloße Funktionalität des Stacks implementiert.
Für die Umsetzung ist zusätzlich wichtig, wie Aktionen gespeichert werden, also mit welchen Daten die Stacks später gefüllt werden. Hierfür bietet sich der Aufzählungsdatentyp *Enum* an, der in C++/CLI zur Verfügung steht. Für jede mögliche Aktion wird ein Eintrag erstellt und diese kann somit nachvollzogen und rückgängig gemacht werden. Bewegungen innerhalb der Ausgabe des Shader Viewers wurden vorerst nicht berücksichtigt, da diese Arbeitsschritte mit den verfügbaren „Reset“ Schaltflächen wieder in ihre Ausgangssituation zurückgesetzt werden können.

## 5.4 Auswahl der richtigen Farben

Im Abschnitt 4.3 wurde bereits kurz auf die Auswahl der richtigen Farbe für die graphische Oberfläche eingegangen. Dort wurde eine graphische Oberfläche gefordert, die auch von Farbenblinden gut bedient werden kann. Um solche Hilfen zu etablieren, gibt es diverse Software Programme. Zusätzliche Überprüfungen sind selbstverständlich hilfreich, da es verschiedene Formen von Farbenblindheit mit unterschiedlicher Stärke und Ausprägung gibt. Ein Tool, das einige der gängigsten Farbschwächen überprüfen kann, ist der „ColorDoctor“ der Firma „Fujitsu“.
Beim ColorDoctor fällt besonders positiv auf, dass mit diesem Programm ein Bereich des sichtbaren Bildschirms ohne Zwischenspeicherung der Bilder analysiert werden kann. Dies ist in Abbildung 30 anhand einer frühen Skizze der graphischen Oberfläche dargestellt. Man sieht deutlich, dass die drei Farbtöne der graphischen Oberfläche auf der linken Seite für einen Normalsichtigen gut zu unterscheiden sind, auf dem Graubild auf der rechten Seite hingegen, ist dies wesentlich schwieriger. Der gelbe und der violette Farbton sind kaum zu unterscheiden. Diese Farbwahl wäre für einen völlig Farbenblinden ungeeignet.
Die unterschiedlich stark ausgeprägten Farbsehschwächen können mit dem ColorDoctor adäquat simuliert werden. Hierfür stehen die folgenden Konvertierungsmöglichkeiten zur Verfügung.

- Graustufen: Völliges Fehlen von Farbe.
- Protanopia: Rotgrünblindheit, Unterscheidung zwischen roten und grünen Farbtönen ist gestört.
- Deuteranopia: Grünblindheit, die Wahrnehmung von grünen Farbtönen ist gestört.

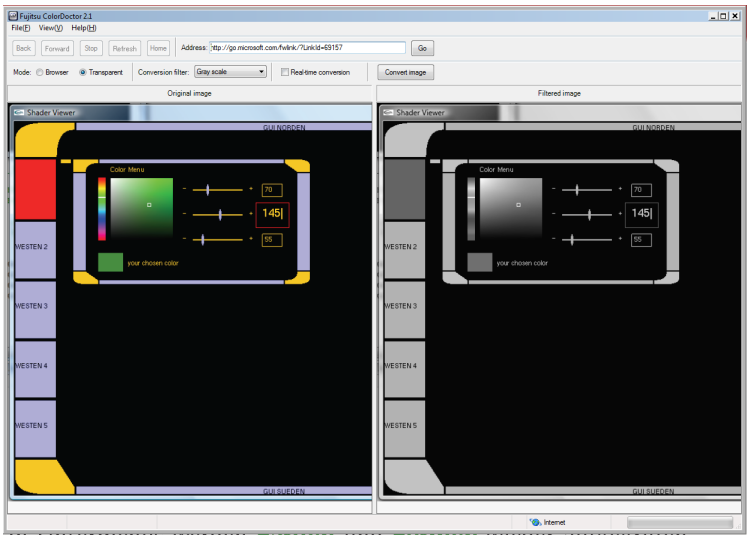


Abbildung 30: Analyse mit *ColorDoctor*

- Tritanopia: Blaublindheit, die Wahrnehmung von blauen Farbtönen ist gestört. [Hun98]

Bevor eine Untersuchung auf Farbkriterien hin durchgeführt werden konnte, musste zunächst eine Auswahl an harmonierenden Farbmustern erstellt werden. Dies geschieht mit Hilfe der Webanwendung „Kuler“ der Firma „Adobe“.
Die Farben können im Kuler in unterschiedlichen Modi direkt über einen Farbkreis ausgewählt werden. Einige dieser Modi werden im Folgenden kurz vorgestellt. Die Einstellung der Farben kann zusätzlich mittels Schiebereglern in verschiedenen Farbräumen wie RGB (Rot, Grün, Blau), HSV (Hue, Saturation, Value) oder CMYK (Cyan, Magenta, Yellow, Key) erfolgen.

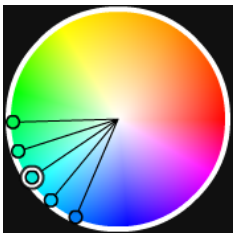

Abbildung 31: Analoge Farbwahl

**Analoge Farbwahl:** Bei der analogen Wahl werden benachbarte Farben, die auf dem Farbkreis dicht beieinander liegen, festgelegt. Die Basisfarbe liegt dabei in der Mitte und die anderen sind rechts und links mit gleichem Abstand davon angeordnet. Der Abstand der Farben von der Basisfarbe kann vergrößert oder verkleinert werden, bleibt jedoch zwischen den einzelnen Komponenten immer konstant. Durch Bewegen der einzelnen Farben in Richtung Kreismittelpunkt kann zusätzlich die Sättigung einzelner oder aller Farben gleichzeitig verändert werden.

Abbildung 32: Monochromatische Farbwahl

**Monochromatische Farbwahl:** Bei dieser Auswahl legt die Basisfarbe den Wert für den Farbton (Hue) fest. Der ermittelte Wert bleibt bei allen anderen Farben konstant. Diese variieren lediglich in ihrer Sättigung und ihrem Hellwert. Dadurch entstehen weiche, balancierte Übergänge, die vom Betrachter meist als angenehm empfunden werden. Allerdings entstehen nur geringe Kontraste und die Farben wirken wenig dynamisch und lebhaft.

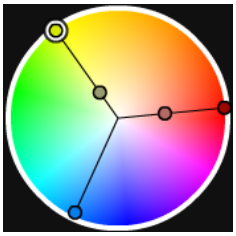

Abbildung 33: Triadische Farbwahl

**Triadische Farbwahl:** Bei der triadischen Auswahl haben drei der gewählten Farben immer maximalen Abstand zueinander, was ihren Farbton (Hue) anbetrifft. Bei voller Sättigung bildet sich also ein gleichseitiges Dreieck auf dem Farbkreis. Die einzelnen Farben können in ihrer Sättigung und ihrem Hellwert modifiziert werden. Es entstehen starke Kontraste, die trotzdem harmonisch wirken.

Abbildung 34: Komplementäre Farbwahl

**Komplementäre Farbwahl:** Bei dieser Auswahl werden zwei Farben ausgewählt, die genau auf den gegenüberliegenden Seiten des Farbkreises angeordnet sind. Auch hier können wieder Hellwert und Sättigung individuell eingestellt werden. Alle Farben befinden sich allerdings auf der Linie, die zwischen den beiden Komplementärfarben aufgespannt wird. Hierdurch entstehen starke Kontraste. Die Wahl einer „warmen“ und einer „kalten“ Farbe führt meist zu den besten Ergebnissen. Warme Farben sind Bunttöne im Bereich des Roten bis Gelben. Sie erzeugen beim Betrachter ein angenehmes, warmes Gefühl. Kalte Farben sind hauptsächlich Blautöne und bläuliche Grüntöne, die beim Betrachter ein Gefühl der Kälte hervorrufen.

Alle Modi verwenden eine kreisförmige Farbdarstellung. Die auf dem äußeren Radius aufgetragenen Farben entsprechen einer Weiterentwicklung des von *Johannes Itten*[22] entwickelten Farbkreises [Itt03]. Zur Mitte hin verlieren die Farben ihre Sättigung, bis sie im Mittelpunkt weiß werden. Der Hellwert der Farbe ist auf einem separaten Regler einstellbar.

[22] Schweizer Maler, Kunsttheoretiker und Kunstpädagoge

Die besten Ergebnisse ließen sich mit der triadischen Wahlmöglichkeit erreichen, da die entwickelte graphische Oberfläche mit drei unterschiedlichen Farben auskommt. Durch die dabei erreichte starke Verteilung der Farbtöne konnten ausreichende Kontraste gewährleistet werden.

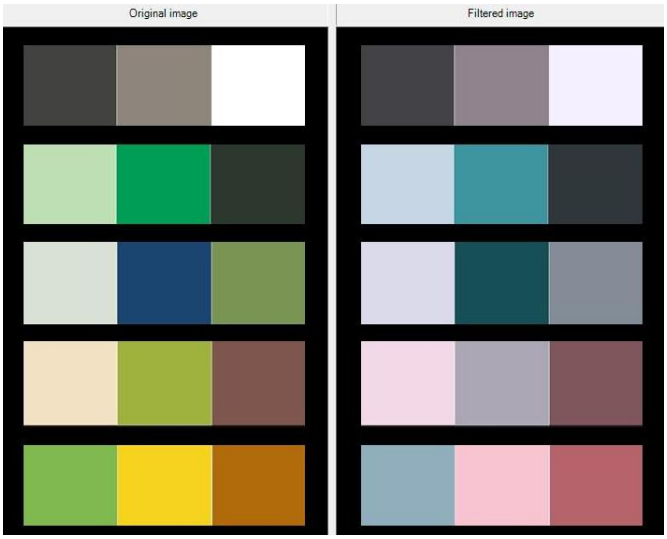


Abbildung 35: Gewählte Farben im Test

Wie in Abschnitt 4.3 bereits angesprochen, sollen die Farben eher unaufdringlich und zurückhaltend wirken, um die Ausgabe des Shader Viewers nicht zu stark zu beeinflussen. Die ausgewählten Farben wurden wiederum darauf geprüft, ob sie von Menschen mit Farbwahrnehmungsschächen unterschieden werden können, wie in Abbildung 35 am Beispiel Tritanopia im Programm ColorDoctor zu sehen ist.
Hatte sich eine Kombination als zweckmäßig erwiesen, wurde sie als auswählbares Farbschema in die graphische Oberfläche aufgenommen.

## 5.5 Evaluierung

Um Schwachstellen in der Usability der graphischen Oberfläche erkennen zu können, sind Evaluierungen unverzichtbar. In Abschnitt 2.2.5 wurden bereits zwei Verfahren vorgestellt. Da heuristische Evaluationen im Optimalfall von drei bis fünf Experten durchgeführt werden, erwies sich dies als schwierig umzusetzen.
Da die Experten parallel und unabhängig voneinander die Evaluation durchführen, können die Heuristiken hier dazu genutzt werden, um eine eigene kurze Überprüfung der graphischen Oberfläche durchzuführen. Dabei werden natürlich wesentlich weniger Fehler gefunden, als beim Test mit mehreren Experten, die mit der graphischen Oberfläche noch nicht so vertraut sind und sich in diese erst einarbeiten müssen. Erste offensichtliche Probleme können durch Eigenanalyse aber durchaus gefunden werden. Die Evaluierung ist schnell und unkompliziert durchzuführen. So wurde die Liste der zehn in Abschnitt 2.2.5 vorgestellten Heuristiken immer wieder an der Software abgearbeitet und getroffene Entscheidungen auf ihre Kongruenz mit den Anforderungen hin überprüft.
Doch selbst wenn es möglich gewesen wäre, eine heuristische Evaluation mit einer ausreichenden Zahl von Experten durchzuführen, kann dies keinen Test mit wirklichen Benutzern

ersetzen.
Schließlich wurde als Kompromiss ein Test mit vier Anwendern durchgeführt, die folgende Aufgaben bearbeiten sollten.

- Laden eines Shaders, Modifizieren eines beliebigen Shader Parameters
- Darstellung eines Torus, Aufbringen einer Textur
- Darstellung eines Würfels, Betrachten von allen Seiten, Bewegen der Lichtquelle hinter das dargestellte Objekt

Da zum Zeitpunkt des Tests noch keine lauffähige Version der graphischen Oberfläche vorlag, musste die Lösung der Aufgaben auf unvollständigen Prototypen durchgeführt werden. Dies erschwerte die Ausführung erheblich. Es konnte kein wirklich zufrieden stellendes Ergebnis ermittelt werden. Da die Prototypen der graphischen Oberfläche z.B. noch keine geeigneten Rückmeldungen auf die Aktionen der Benutzer geben konnten, gab es viele Zwischenfragen der Testanwender, die den Arbeitsablauf störten.
Die Ausführung einer Evaluierung an einem Prototypen war problematischer als erwartet. Die gestellten Aufgaben waren wahrscheinlich nicht an den Entwicklungsstand der graphischen Oberfläche angepasst und damit zu komplex und schwierig.
Trotz aller Widrigkeiten brachte die Evaluierung dennoch einige verwertbare, konstruktive Ergebnisse, nachdem die Abarbeitung der ursprünglichen Aufgaben abgebrochen worden war und ein weniger ambitioniertes Prüfungsschema verwendet wurde. Die Testbenutzer sollten während der Verwendung der Software „laut denken“ (Think Aloud). Hierbei erzählt der Testbenutzer, was ihm gerade an der graphischen Oberfläche auffällt, stört oder was genau ihm momentan Probleme bereitet.
Hierbei gab es vor allem Kritik an der Farbgebung der graphischen Oberfläche. Teilweise wurde auch die Größe der Schaltflächen bemängelt, da diese die Ausgabe des Shaders zu sehr einengen.
Somit konnten doch noch ein paar Verbesserungsvorschläge aus dem Test mit in die Entwicklung einfließen. Um wirklich repräsentative Ergebnisse zu erhalten, wären allerdings mehr als vier Anwender bei der Software-Prüfung notwendig gewesen.

# 6 Resümee, Fazit

Im Folgenden werden die in dieser Arbeit erreichten Ergebnisse und die gewonnenen Erkenntnisse diskutiert und ein Ausblick für zukünftige Entwicklungen gegeben.

## 6.1 Ergebnisse

Ziel dieser Arbeit war es, eine graphische Oberfläche für einen Shader Viewer zu entwickeln. Ein weiterer Fokus lag auf der konsequenten Umsetzung der HCI Vorgaben. Abweichungen von diesen Richtlinien wurden begründet und kritisch geprüft.
Das Einhalten von HCI Richtlinien allein garantiert allerdings noch nicht die Entwicklung einer geeigneten, benutzerfreundlichen graphischen Oberfläche. Es gab viele weitere projekt- und situationsspezifische Anforderungen, die mit aufgenommen und berücksichtigt werden mussten. Bei der Umsetzung wurde in dieser Arbeit der Versuch unternommen, möglichst viele dieser individuellen Aspekte zu berücksichtigen und bei der Entwicklung der Software ein besonderes Augenmerk auf die vermuteten Bedürfnisse des Anwenders zu legen.
Zur Überprüfung der gefundenen Lösungen wären allerdings noch weitergehende Evaluierungen notwendig gewesen, die im Rahmen dieses Projekts jedoch nicht realisiert werden konnten. Auch müsste noch abgewartet werden, welchen Anklang und welche Akzeptanz das fertige Softwareprodukt mit seiner graphischer Oberfläche in der alltäglichen Praxis beim Gros der Anwender finden würde. Ohne diese abschließenden Feldversuche ist es schwierig, die Güte und die Qualität der gefunden Lösung zu beurteilen.
In dieser Arbeit wurde ein weiterer Schwerpunkt auf die Möglichkeit der Verwendung von Guidelines und den darin empfohlenen standardisierten Steuerelementen gelegt. Deren Vor- und Nachteile wurden aufgezeigt und dem eigenen Ansatz gegenübergestellt. Ob sich der Mehraufwand gelohnt hat, auf diese vorgegebenen Frameworks zu verzichten, lässt sich nur schwer und nicht eindeutig beantworten. Dies liegt wiederum hauptsächlich an fehlenden Informationen aus den noch ausstehenden abschließenden Evaluationen unter Praxisbedingungen. Zusätzlich hätten die selbstentwickelten Ansätze mit anderen Lösungen, die mit ähnlichem Arbeitsaufwand unter Verwendung von Standardelementen erzielbar gewesen wären, verglichen werden müssen. Die eingesparte Zeit gegenüber der Eigenentwicklungen von Steuerelementen hätte beispielsweise in weiterreichende Funktionen, in das optimale Zusammenwirken der einzelnen Elemente oder in eine innovativere Struktur investiert werden können.
Während der Ausarbeitung stellte sich die Zusammenarbeit mit dem Entwickler des Shader Viewers als vorteilhaft für den Entwurf und die Implementierung der graphischen Oberfläche heraus. Durch Rücksprachen konnten gewisse Ideen mit in die graphische Oberfläche einfließen. Zusätzlich erleichterten vereinbarte Anpassungen der Schnittstelle zwischen Programm und graphischer Oberfläche dessen Entwicklung erheblich. Nur wenn die Intention der zugrun-

de liegenden Software richtig verstanden ist, kann auch eine möglichst optimale graphische Oberfläche erstellt werden.
Mit den in dieser Arbeit durchgeführten Analysen und dem daraus resultierenden mäßigen Abschneiden von Produkten mit ähnlicher Funktionalität, wie sie der Shader Viewer aufweist, wurde gezeigt, dass eine solche Software durchaus sinnvoll ist und weiterentwickelt werden sollte.

## 6.2 Ausblick

Generell ließen sich durch zusätzliche Evaluierungen und Anwendungerfahrungen am fertigen Produkt noch unerkannte Schwachstellen und Probleme der graphischen Oberfläche beseitigen. Vor allem bei den angebotenen Hilfefunktionen besteht noch erhebliches Optimierungspotential. Durch Nachbesserungen der in den Evaluierungen aufgefundenen Probleme könnte ein verbessertes und leichter zu bedienendes Produkt entstehen.
Auch umfangreiche, generelle Erweiterungen sind denkbar. Hierbei läge der Fokus nicht nur auf der graphische Oberfläche, sondern zwangsläufig müsste auch das zugrunde liegende Programm mit einbezogen werden.
Der Shader Viewer ist derzeit ein Programm zum Betrachten von Shader Effekten. Würde er jedoch um einen Editor erweitert, könnte eine integrierte Entwicklungsumgebung entstehen. Der Funktionsumfang der Software würde eine erhebliche Erweiterung erfahren und die graphische Oberfläche müsste natürlich dementsprechend angepasst werden.
Eine größere Funktionalität bedeutet meist auch eine komplexere Schnittstelle mit dem Benutzer. Elemente der graphischen Oberfläche müssten an die neuen Anforderungen angepasst werden, um weiterhin bedienerfreundlich und zielorientiert mit dem Anwender kommunizieren zu können.
Natürlich sind auch begrenztere Erweiterungen denkbar, die lediglich die graphische Oberfläche betreffen. Hier wäre beispielsweise die Kommunikation mit dem Dateisystem zu erwähnen. Drag and Drop Operationen zwischen Fenstern des Betriebssystems und dem Shader Viewer wären eine zusätzlich mögliche Erweiterung.
Sinnvoll, effektiv und einfach sind die hier diskutierten Erweiterungsmöglichkeiten vor allem dann umzusetzen, wenn ein akuter Bedarf nach Rückmeldungen und Erfahrungsberichten von Benutzern vorliegt. Ansonsten besteht die Gefahr, an den Wünschen und Anforderungen der Anwender „vorbei zu entwickeln“. Oft stellt sich Verbesserungsbedarf auch erst nach längeren Phasen der Verwendung der Software heraus.

# Literatur


[App08a] Apple Inc.: *Apple Human Interface Guidelines; User Experience.* 2008-06-09. Cupertino, California, 2008. `http://developer.apple.com/documentation/userexperience/Conceptual/AppleHIGuidelines/OSXHIGuidelines.pdf`. – Verfügbar online unter `http://developer.apple.com/documentation/userexperience/Conceptual/AppleHIGuidelines/`; zuletzt besucht am 6. März 2009

[App08b] Apple Inc.: *Apple Publications Style Guide; September 2008.* 2008 edition. Cupertino, California, 2008. `http://developer.apple.com/documentation/userexperience/Conceptual/APStyleGuide/APSG_2008.pdf`. – Verfügbar online unter `http://developer.apple.com/documentation/userexperience/Conceptual/AppleHIGuidelines/`; zuletzt besucht am 6. März 2009

[BCW01] Benyon, David ; Crerar, Alison ; Wilkinson, Simon: Individual Differences and Inclusive Design. In: Stephanidis, Constantine (Hrsg.): *User Interfaces for All; Concepts, Methods and Tools.* Mahwah, New Jersey : Lawrence Erlbaum Assoc Inc, 2001. – ISBN 0–8058–2967–9, S. 21–46

[Cog99] Cognetics Corporation: *The LUCID Design Framework (Logical User Centered Interaction Design).* Website. `ftp://ftp.cs.umanitoba.ca/pub/cs371/Readings/Lucid2a-overview.pdf`. Version: 1999. – Verfügbar online unter `http://www.cognetics.com/`; zuletzt besucht am 6. März 2009

[DFAB98] Dix, Alan J. ; Finlay, Janet E. ; Abowd, Gregory D. ; Beale, Russel: *Human-Computer Interaction.* 2. Aufl. Essex : Prentice Hall, 1998. – ISBN 0–13–239864–8

[FK03] Fernando, Randima ; Kilgard, Mark J.: *The Cg Tutorial; The Definitive Guide to Programmable Real-time Graphics.* Amsterdam : Addison-Wesley Longman, 2003. – ISBN 0–321–19496–9

[HBC+92] Hewett, Thomas T. ; Baecker, Ronald ; Card, Stuart ; Carey, Tom ; Gasen, Jean ; Mantei, Marilyn ; Perlman, Gary ; Strong, Gary ; Verplank, William ; Hefley, Bill (Hrsg.): *ACM SIGCHI Curricula for Human-Computer Interaction.* New York : Association for Computing Machinery, Inc., 1992. – ISBN 0–89791–474–0

[Hun98] Hunt, R.W.G.: *Measuring Colour.* 3. Aufl. Kingston-upon-Thames : Fountain Press, 1998. – ISBN 0–86343–387–1

[Itt03] Itten, Johannes: *Kunst der Farbe, Studienausgabe.* 28. Aufl. Freiburg : Christophorus-Verlag, 2003. – ISBN 3–332–01470–6

[Joh00] Johnson, Jeff: *GUI Bloopers; Don'ts and Do's for Software Developers and Web Designers.* 1. Aufl. Morgan Kaufmann, 2000. – ISBN 1–55860–582–7

[Joh07] Johnson, Jeff: *GUI Bloopers 2.0; Common User Interface Design Don'ts and Dos.* 2. Aufl. Elsevier Ltd, 2007. – ISBN 0–12–370643–2

[Kil96] Kilgard, Mark J.: *OpenGL Programming for the X Window System*. Amsterdam : Addison-Wesley Longman, 1996. – ISBN 0–201–48359–9

[Krö08] Krömker, Detlef: *Grundlagen der Computergraphik*. Frankfurt : Vorlesung Sommersemester, Professur für Graphische Datenverarbeitung am Fachbereich Informatik und Mathematik der Goethe-Universität, 2008

[Mic08] Microsoft Corporation: *Windows User Experience Interaction Guidelines*. November 6, 2008 edition. Redmond, Washington, 2008. `http://download.microsoft.com/download/e/1/9/e191fd8c-bce8-4dba-a9d5-2d4e3f3ec1d3/ux%20guide.pdf`. – Verfügbar online unter `http://msdn.microsoft.com/en-us/library/aa511258.aspx`; zuletzt besucht am 6. März 2009

[Nie93] Nielsen, Jakob: *Usability Engineering*. San Diego : ACADEMIC PRESS, 1993. – ISBN 0–12–518406–9

[Ros06] Rost, Randi J.: *The OpenGL Shading Language*. 2. Aufl. Amsterdam : Addison-Wesley Longman, 2006. – ISBN 0–321–33489–2

[RP90] Rock, Irvin ; Palmer, Stephen: The Legacy of Gestalt Psychology. In: *Scientific American* v263 n6 p84-90 (1990), Dezember

[Sch05] Schwichtenberger, Holger: *Microsoft .net2.0 Crashkurs*. Unterschleißheim : Microsoft Press Deutschland, 2005. – ISBN 3–86063–531–X

[SM00] Schumann, Heidrun ; Müller, Wolfgang: *Visualisierung, Grundlagen und allgemeine Methoden*. Berlin Heidelberg : Springer-Verlag, 2000. – ISBN 3–540–64944–1

[Thi00] Thissen, Frank: *Screen-Design-Handbuch*. Springer-Verlag GmbH, 2000. – ISBN 3–540–64804–6

[Wat99] Watt, Alan: *3D Computer Graphics*. 3. Aufl. Amsterdam : Addison-Wesley Longman, 1999. – ISBN 0–201–39855–9

[WNDS99] Woo, Mason ; Neider, Jackie ; Davis, Tom ; Schreiner, Dave: *OpenGL Programming Guide*. 3. Aufl. Boston : Addison-Wesley, 1999. – ISBN 0–201–60458–2