TUHH
Technische Universität Hamburg-Harburg

Diplomarbeit

Entwicklung eines Autorentools für ein sphärisches Projektionssystem

vorgelegt von:	Dmitrijs Barbarins
Fachbereich:	Informatik-Ingenieurwesen
Matrikelnummer:	22273
Erstgutachter:	Dr. rer.-nat. habil. Ralf Möller
Zweitgutachter:	Dr. Helmut Weberpals

Fraunhofer Verbund Informations- und Kommunikationstechnik

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

Entwicklung eines Autorentools für ein sphärisches Projektionssystem

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Diplomarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der TUHH abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Berlin, den 14. Februar 2009

_____

Dmitrijs Barbarins

Inhaltsverzeichnis

Eidesstattliche Erklärung I

Abbildungsverzeichnis VII

Tabellenverzeichnis IX

Verzeichnis der Listings XI

1 Einleitung 1

1.1 Aufbau der Arbeit . . . 1
1.2 Richtlinien . . . 2
1.3 Die Fraunhofer-Gesellschaft . . . 3
1.4 Aufgabenstellung . . . 4

2 Thematische Einführung 5

2.1 Bedeutung der Visualisierung . . . 5
2.2 Visualisierungsmedien . . . 7
2.2.1 3D Displays . . . 7
2.2.2 Digitalprojektoren . . . 8
2.2.3 3D-Projektionssysteme . . . 9
2.3 Sphärisches Projektionssystem . . . 10

3 Technisches Umfeld 13

3.1 Aufbau und Funktion des Projektionssystems . . . 13
3.1.1 Softwarekomponenten . . . 15
3.1.2 3D-Rendering-Engine . . . 16
3.2 Aufbau der Szene . . . 18
3.2.1 Struktur der Szenen für den Omniglobe . . . 20
3.2.2 Scene-XML-Konfigurationsdateien . . . 21
3.2.3 Skripten . . . 22

3.2.4 Effekte . . . 24
3.3 Benutzte Programmbibliotheken und Programmiertechniken . . . 26
3.3.1 Grafische Benutzeroberfläche . . . 28
3.3.2 Loggen . . . 33
3.3.3 XML . . . 35
3.3.4 Signal-Slot-Mechanismus . . . 35
3.4 Entwicklungstools . . . 36
3.4.1 Versionsverwaltung . . . 36
3.4.2 Dokumentation . . . 38

4 Planung **39**
4.1 Voranalyse und Anforderungen . . . 39
4.1.1 Design Varianten . . . 40
4.1.2 Entwicklungsumgebung und Programmiersprache . . . 41
4.2 Pflichtenheft . . . 42
4.2.1 Zielbestimmungen . . . 42
4.2.2 Einsatz . . . 45
4.2.3 Produktumgebung . . . 46
4.2.4 Produktfunktionen . . . 47
4.2.5 Produktdaten . . . 50
4.2.6 Produktleistungen . . . 51
4.2.7 Benutzungsoberfläche . . . 51
4.2.8 Qualitätszielbestimmungen . . . 54
4.2.9 Globale Testszenarien und Testfälle . . . 54
4.3 Abschätzungen . . . 55

5 Implementierung **59**
5.1 Lösungsweg . . . 62
5.1.1 Realvorschau- und Bearbeitungsmodus . . . 62
5.1.2 Umstrukturierung der Szene . . . 64
5.1.3 Implementierung der HLSL Effekte . . . 67
5.2 Netzwerkschicht . . . 68
5.3 Benutzeroberflächenschicht . . . 72
5.3.1 Parameter Frame . . . 77
5.4 Dataschicht . . . 79
5.4.1 Erstellen einer Szene . . . 80
5.4.2 Export und Import der Projektdatei . . . 83

6 Bewertung und Ausblick **87**
6.1 Tests 87
6.2 Ergebnisse 89
6.3 Vergleich vorher/nachher 90
6.4 Bewertung 91
6.5 Weiterentwicklung des Systems 92

7 Abschließende Betrachtung **95**
7.1 Erfahrungsbericht 95
7.2 Zusammenfassung 96

Literaturverzeichnis **97**

A Anhang **i**

Abbildungsverzeichnis

2.1 Das “Liquid Browsing”-Konzept. 6
2.2 VR-Object-Display des Fraunhofer FIRST Instituts 9
2.3 “Mixed-Reality”-Display 9
2.4 Der Omniglobe. 11

3.1 Schematische Darstellung des Omniglobe 15
3.2 Überschneidung von Bildern beider Projektoren 15
3.3 Eingangsbild 16
3.4 Die resultierenden Bilder für Projektoren 16
3.5 Aufbau einer Szene 20
3.6 Glade “Interface Builder” 31

5.1 Systemarchitektur, logische Aufteilung 61
5.2 Bearbeitungsmodus 63
5.3 Realvorschaumodus 63
5.4 Aufrufe zur build-Methode 65
5.5 Struktur der Szene 66
5.6 Klassendiagramm der Netzwerkschicht 69
5.7 Benutzeroberfläche des Autorentools 74
5.8 Verlauf der Signale 75
5.9 Klassendiagramm der Benutzeroberflächenschicht 76
5.10 Klassendiagramm der Parameter Frames 78
5.11 Klassendiagramm der Dataschicht 81
5.12 Lebenszyklus des Autorentools 82
5.13 Callgraph der build Methode 84

6.1 “Presentation”-Szene im Bearbeitungsmodus 89
6.2 “Presentation”-Szene auf dem Omniglobe 89

Tabellenverzeichnis

3.1 Typen der Ereignisse für Rückruffunktionen 24

4.1 Qualitätszielbestimmungen 54
4.2 Abschätzung der Umsetzbarkeit 57

5.1 Vergliech der Produktfunktionen 64

6.1 Vergleich vorher/nachher 91

Verzeichnis der Listings

3.1 Beispiel der *devdelight.xml*-Konfigurationsdatei 19
3.2 Beispiel des ScriptCallbacks 23
3.3 Beispiel des Init-Skriptes 25
3.4 Beispiel des Pixel-Shaders 27
3.5 Einsatz der-Libglade Bibliothek. 32
3.6 Einsatz der EZLogger-Bibliothek. 34
3.7 Beispiel der Signal-Slot-Technik 37
3.8 Beispiel der Doxygen-Kommentare 38
5.1 Beispiel der Template-Singleton-Technik 62
5.2 Beispiel der Netzwerkkommunikation 71
5.3 Beispiel der Projektdatei 85
A.1 Beispiel der Szene XML-Konfigurationsdatei i

Kapitel 1

Einleitung

Im Rahmen der vorliegenden Diplomarbeit wurde ein Autorentool für ein sphärisches Projektionssystem entwickelt. Anhand dieses Softwaresystems sollen die Contents in Form der Szenen zur Darstellung mithilfe des Projektionssystems entwickelt werden.

1.1 Aufbau der Arbeit

Diese Abhandlung umfasst alle Phasen der Softwareentwicklung von der formalen Anforderungsdefinition und Spezifikation über den Entwurf sowie die Implementierung bis hin zum Test und zur Dokumentation.

Kapitel 1, Einleitung, gibt eine allgemeine Vorstellung von der Ausführung und den Richtlinien der Arbeit sowie die Aufgabenstellung aus der Sicht des Auftraggebers. Die Aufgabe wird hier sehr allgemein definiert, eine ausführliche formale Beschreibung folgt in Kapitel 4, Planung.

Kapitel 2, Thematische Einführung, führt allgemein in die Thematik und das fachliche Umfeld der Arbeit ein.

Kapitel 3, Technisches Umfeld, präsentiert das technische Umfeld. Dabei werden die technische Ansätze des Projektionssystems und die grundlegenden Eigenschaften der eingesetzten standardisierten Technologien und Frameworks sowie ihre Zusammenhänge erläutert.

Kapitel 4, Planung, gibt die Definitionen der Anforderungen an das System formal und

detailliert in Form eines Pflichtenheftes sowie einige Abschätzungen des entwickelten Systems.

Kapitel 5, Implementierung, beschreibt aufbauend auf die Anforderungen des Pflichtenheftes das realisierte Softwaresystem.

Kapitel 6, Bewertung und Ausblick, fasst die Ergebnisse der Abhandlung zusammen. Hier werden die realisierten Funktionen analysiert, Testergebnisse vorgestellt sowie die Ideen für die neu zu implementierenden Funktionen des Systems präsentiert. Ferner erfolgt eine Bewertung des entwickelten Systems.

Kapitel 7, Abschließende Betrachtungen, beschreibt die bei der Entwicklung gesammelten Erfahrungen und fasst die Arbeit zusammen.

1.2 Richtlinien

Zur besseren Orientierung werden folgende Richtlinien bei der Formatierung des Textes eingehalten. Diese Richtlinien beziehen sich auf die Auszeichnung von Zitaten, Bezeichnungen und Begriffe.

Zur besseren Orientierung werden folgende Richtlinien bei der Formatierung des Textes eingehalten. Diese Richtlinien beziehen sich auf die Auszeichnung von Zitaten, Bezeichnungen und beschriebenen Begriffen (vgl Autorname):

> *"Der Text der direkten Zitate hat dieses Format. Direkte Zitate wurden wörtlich aus der Originalquelle übernommen. Die Quellenangabe aus der Literaturliste steht vor dem Zitat."*

Indirekte Zitate werden mit einem Verweis auf benutzte oder weiterführende Literatur verwendet. Sie geben den Inhalt der Originalquelle nicht wörtlich, sondern sinngemäß wieder. Die Zitate werden nicht durch eine besondere Formatierung gekennzeichnet, besitzen aber ebenfalls eine Quellenangabe.

Der Quellcode des Systems wird in der Arbeit lediglich in sehr kleinen Ausschnitten präsentiert. Die vollständige Version befindet sich auf der beigelegten CD. Die Quellcode-Beispiele können sich unwesentlich von der Implementierung unterscheiden. Dies dient dem besseren und leichteren Einblick in die Programmlogik. Die Style-Guides der Beispiele wurden ebenfalls etwas verändert, damit eine möglichst kompakte Darstellung entsteht.

So werden beispielsweise Blockklammern in der ersten bzw. letzten Zeile des Blocks platziert.

1.3 Die Fraunhofer-Gesellschaft

Die Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. ist eine Organisation für angewandte Forschungs- und Entwicklungsdienstleistungen in Europa. Sie besteht aus 56 Forschungsinstituten, die in sieben Verbünde eingeteilt sind:

- Informations- und Kommunikationstechnik
- Life Sciences
- Mikroelektronik
- Oberflächentechnik und Photonik
- Produktion
- Verteidigungs- und Sicherheitsforschung
- Werkstoffe, Bauteile

Der Fraunhofer-Verbund Informations- und Kommunikationstechnik (nachfolgend IuK-Verbund) besteht aus 15 Mitgliedsinstituten, die im Bereich Informations- und Kommunikationstechnik angesiedelt sind. Auf Verbundsebene sind die Tätigkeiten der Institute in zehn Geschäftsfelder zusammengefasst:

- digitale Medien
- E-Business
- E-Government
- Kommunikationssysteme
- Kultur und Unterhaltung
- Medizin und Life Sciences
- Produktion
- Security
- Software

- Verkehr und Mobilität

Die einzelnen Institute können auch in mehreren Geschäftsfeldern tätig sein.

1.4 Aufgabenstellung

Für das bereits existierende Projektionssystem wird ein Autorentool im Zusammenspiel mit eingesetzten Hardware- und Softwarekomponenten des Projektionssystems entwickelt.

Das Autorentool soll die intuitive Erstellung von Szenen (Contents) ermöglichen und die am häufigsten benutzten Vorgänge automatisieren.

Anhand des Autorentools können folgende Elemente als Teilkomponenten der Szene addiert und entsprechend bearbeitet werden:

- Modelle
- Ebenen
- Linien
- Texturen
- Videos
- HLSL-Effekte
- Texte

Zur leichteren Bearbeitung wird die Szene als Realvorschau angezeigt und bei Bedarf im Erstellungsprozess auf dem Omniglobe dargestellt.

Das Ergebnis der Arbeit bilder eine vollständige Szene, die man - nach Wunsch des Benutzers - automatisch auf den realen Omniglobe übertragen kann.

Kapitel 2

Thematische Einführung

Nach der allgemeinen Einleitung in Kapitel 1 soll in diesem Kapitel nun eine Einführung in die thematischen Grundlagen dieser Diplomarbeit geliefert werden. Zu diesem Zweck wird in den Abschnitten 2.1 und 2.2 die Bedeutung der Visualisierung und eine Einleitung in die Entwicklung der Visualisierungsmedien gegeben. Im Anschluss präsentiert Abschnitt 2.3 das sphärische Projektionssystem.

2.1 Bedeutung der Visualisierung

Die Darstellung und Veranschaulichung verschiedenster Informationen war stets von großer Bedeutung. Die Entwicklung der Technik und insbesondere der Informationstechnik eröffnete neue Möglichkeiten sowie Bedürfnisse in diesem Bereich. Die ersten und grundsätzlichen Aufgaben der Informatik bestanden in der Speicherung und Verwaltung von Informationen. Diese Aufgaben wurden - von einigen Sackgassen abgesehen - effektiv gelöst. Andererseits führte die Entstehung neuer Informationstechnologien unter anderem zu einer Informationsexplosion. Laut einer Studie von Peter Lyman und Hai R. Varian[1] erhöhte sich das Wachstum an gespeicherten Informationen zwischen 1999 und 2002 jährlich um 30 Prozent.

Als Folge entstand ein neues Forschungsgebiet, das sich mit dem Thema "Darstellung und Visualisierung der Informationen" beschäftigt. Neue Visualisierungsmethoden bieten eine Hilfestellung bei den Versuchen, größere Mengen von Informationen leichter und kom-

[1] Vgl. [Lyman und Varian 2003]

pakter zu veranschaulichen. Beispielsweise wurde ein sogenanntes Liquid Browsing-Konzept[2] des Dateimanagers vorgestellt. "Liquid Browsing" setzt auf sogenannte Scatter Graphs. Diese repräsentieren jedes Objekt durch einen Punkt in einem zweidimensionalen Koordinatensystem. Jede Achse spiegelt eine Eigenschaft eines Objekts wider. Als dritte Variable gibt die Punktgröße eine weitere Objekteigenschaft wieder, weitere Parameter lassen sich durch die Deckkraft oder den Farbwert der Punkte darstellen. Eine der wichtigsten Eigenschaften dieses Konzepts bildet das flüssige und interaktive Verhalten im Rahmen von Benutzeraktivitäten.

Als weitere Beispiele sind anzuführen:

- Gapminder: http://graphs.gapminder.org/world/

- IBM Manyeyes: http://manyeyes.alphaworks.ibm.com/manyeyes/

Abbildung 2.1: Das "Liquid Browsing"-Konzept.

[2] Wird auch "Liquifile" genannt. Siehe http://www.liquifile.info/ zu mehr Informationen über dieses Konzept.

2.2 Visualisierungsmedien

Mit der wachsenden Bedeutung der Visualisierungsmethoden wurden gleichermaßen die Visualisierungsmedien verbessert. Die Entwicklung der Technik führte zu deutlichen Verbesserung der bekannten und weitverbreiteten Medien, wie z. B. Bildschirme. Bildschirme fungieren noch immer als die am häufigsten eingesetzten Medien, was ihrem niedrigen Preis, ihrer guten Bildqualität und Anschaubarkeit geschuldet ist. Die einzelnen Nachteile oder eher verständliche Begrenzungen von Medien dieses Typs sind eine niedrige Skalierbarkeit sowie die reine zweidimensionale Natur des Bildes. Und genau an diesem Punkt setzen mehrere Forschungsgruppen rund um die Welt an.

2.2.1 3D Displays

Dreidimensionale Bildschirme bilden eine der Entwicklungen konventioneller Bildschirme. Es finden sich mehrere Versuche, die zweidimensionale Begrenzung zu überwinden, wobei die Grundidee der Stereoskopie für die meisten Technologien identisch ist.

Damit ein räumlicher Eindruck entsteht, erzeugt eine spezielle Software von jedem Objekt zwei unterschiedliche Bilder - eines für das rechte und eines für das linke Auge. Die Technologien unterscheiden sich meistens in der Art und Weise, wie die zwei Bilder in Bezug auf den Benutzer visualisiert werden. Sämtliche Technologien kann man in folgende Gruppen unterteilen:

Stereoskopische 3D Displays Diese umfassen u. a. Head-Mounted-Display, Polarisationsfilter, Anaglyphenbrille, Shutterbrille.

Autostereoskopische 3D Displays Autostereoskopische Displays oder 3D-Monitore sind in der Lage, stereoskopische oder dreidimensional erscheinende Bilder anzuzeigen, ohne dass sich der Benutzer weiterer Hilfsmittel, wie etwa Brillen, bedienen muss.

Der vom Fraunhofer Heinrich-Hertz-Institut entwickelte Bildschirm “Mixed-Reality Display” beispielsweise ist ein Single-User-3D-Display. Bei diesem Display werden die Bilder zweier hochaufgelöster Displaypanel - jedes mit 1600 x 1200 RGB-Pixeln - durch eine 30 Zoll große Filterscheibe projiziert. Zu diesem Zweck werden die Bilder für das linke und das rechte Auge des Benutzers in feine senkrechte Streifen unterteilt und abwechselnd nebeneinander angeordnet. Um die Bildinformationen optisch richtig zu adressieren, kommt

z. B. eine Speziallinse vor dem Display zum Einsatz. Die gekrümmte Linsenoberfläche leitet das Licht des Bildschirms in zwei Richtungen. Die für das rechte Auge bestimmten Strahlen werden leicht nach rechts verschoben, die für das linke Auge leicht nach links. Damit der Betrachter auch die richtigen Bilder für das jeweilige Auge erhält, erfasst eine Kamera am Display fortwährend die Position des Kopfes. Bewegt sich der Nutzer, werden die Bildstreifen automatisch versetzt und an den veränderten Blickwinkel angepasst. Die erzeugten Modelle schweben dabei ähnlich wie ein Hologramm frei vor dem Display. Solche Displays nennt man Single-User-Displays, da sich aus oben genannten Gründen lediglich eine Person im sogenannten Sweet Spot[3] befindet.

Bei einer Weiterentwicklung dieser Technologie, sogenannten Multi-User-Displays, versucht man, mehrere Sweet Spots gleichzeitig zu erzeugen. Dafür werden komplexere Linsenoberflächen so eingesetzt, dass mehrere Bilder ausgestrahlt werden. Dadurch erhöht sich die Bewegungsfreiheit, und die Bildqualität sinkt.

Aktuell existieren verfügbare Lösungen von Philips, NewSight, Pavonine, Spatial View oder Fraunhofer HHI mit verschiedenen technischen Vor- und Nachteilen etwa hinsichtlich Auflösung, Bildqualität und Bewegungsfreiheit.

2.2.2 Digitalprojektoren

Eine andere Art der Visualisierungsmedien wurde aus Diaprojektoren entwickelt. Digitalprojektoren projizieren Daten aus einem visuellen Ausgabegerät für ein Publikum in vergrößerter Form an eine Projektionsfläche und werden dementsprechend hauptsächlich bei Präsentationssystemen eingesetzt. Mithilfe von Digitalprojektoren wird das Skalierbarkeitsproblem der konventionellen Displays zum Teil gelöst. Da die Digitalprojektoren auf dieselben Displaytechnologien, wie z. B. LCD, basieren, bleibt die Auflösung von Digitalprojektoren genauso gering wie die Auflösung konventioneller Displays. Zur Generierung einer viel höheren Auflösung kommen mehrere Projektoren (Tiled Display) zur Erstellung einer Projektion gleichzeitig zum Einsatz. Dieses Prinzip wurde bei der Realisierung des HEyeWall-Systems verfolgt, das erstmals auf der CeBIT 2004 einer breiten Öffentlichkeit vorgestellt wurde. Das Grundprinzip eines Tiled Displays besteht darin, mehrere Projektoren in einem Raster anzuordnen. Jeder einzelne Projektor projiziert nur einen Teil des Gesamtbildes. Dabei entstehen Probleme wie etwa Edge Blending, Farbkalibrierung und

[3] Sweet Spot ist die optimale Position, an der sich der Kopf des Benutzers befinden muss, um einen korrekten Stereoeindruck zu bekommen.

Anwendungsverteilung. Die Installation in Darmstadt hat eine Projektionsfläche von 4,8 m auf 2,4 m, besteht aus 48 Projektoren und unterstützt die stereoskopische Ausgabe. Das Projektionssystem wird von 48 Standard-PCs mit NVidia-Grafikhardware betrieben.

2.2.3 3D-Projektionssysteme

Bisherige Produktpräsentationen in 3D hinterlassen stets den Eindruck, das 3D-Objekt sei in eine Art Schaufenster gebannt. Aus unterschiedlichen Blickwinkeln kann man das Objekt nur mithilfe spezieller Interaktionstechniken, wie z. B. Joysticks, betrachten, was eine Distanz zum Objekt erzeugt. Vor diesem Hintergrund hat Fraunhofer FIRST eine digitale Litfasssäule entwickelt, die neben der Darstellung von Standbildern oder Videos auch Panoramen und Objekte in 3D realitätsgetreu projizieren kann. Der Prototyp wurde in Form eines Zylinders realisiert. Er ist mit acht Standardprojektoren ausgestattet, die das Objekt auf eine Spezialleinwand projizieren. Dabei wird das Licht der Projektoren mit dem sogenannten Infitec-Verfahren gefiltert, das heißt, das Licht wird in drei schmalbandige Kanäle (rot, grün, blau) gesplittet, woraus ein sehr sattes und kontrastreiches Bild resultiert. Anhand dieses Verfahrens kann eine Auflösung von rund 2000 x 500 Pixel für die 3D-Projektion erreicht werden. Die Projektoren werfen ihr Licht nicht direkt auf die Leinwand des Zylinders, sondern über Spiegel. Um bei 3D-Darstellungen einen optimalen Bildeindruck zu erreichen, wird ein Trackingmodus eingesetzt. Über einen magnetischen Tracker werden die genaue Position und die Blickrichtung des Betrachters ermittelt. Die Trackingsoftware richtet das Objekt dann entsprechend aus.

Abbildung 2.2: VR-Object-Display des Fraunhofer FIRST Instituts

Abbildung 2.3: “Mixed-Reality”-Display

2.3 Sphärisches Projektionssystem

In Rahmen dieser Arbeit wird ein ganz neues und innovatives Medium analysiert. Der Omniglobe ist ein Zusammenspiel von computergesteuerter Spezialsoftware mit komplexer Technik, das ein von allen Seiten betrachtbares Kugelprojektionssystem bildet. Das System kann beispielsweise die Erde komplett darstellen - bewegt sowie inklusive Wetter- und Standortinformationen und mit Videoeinblendungen.

Der Omniglobe ist ein vielseitig einsetzbares Medium. Neben dem Zweck der Darstellung von Geoinformationen jeglicher Art findet der Omniglobe auch Anwendung als perfektes Präsentationsmedium, z. B. für Messen, Museen sowie in der Werbebranche und im Showbereich. Die weiteren technischen Details sind Kapitel 3 zu entnehmen.

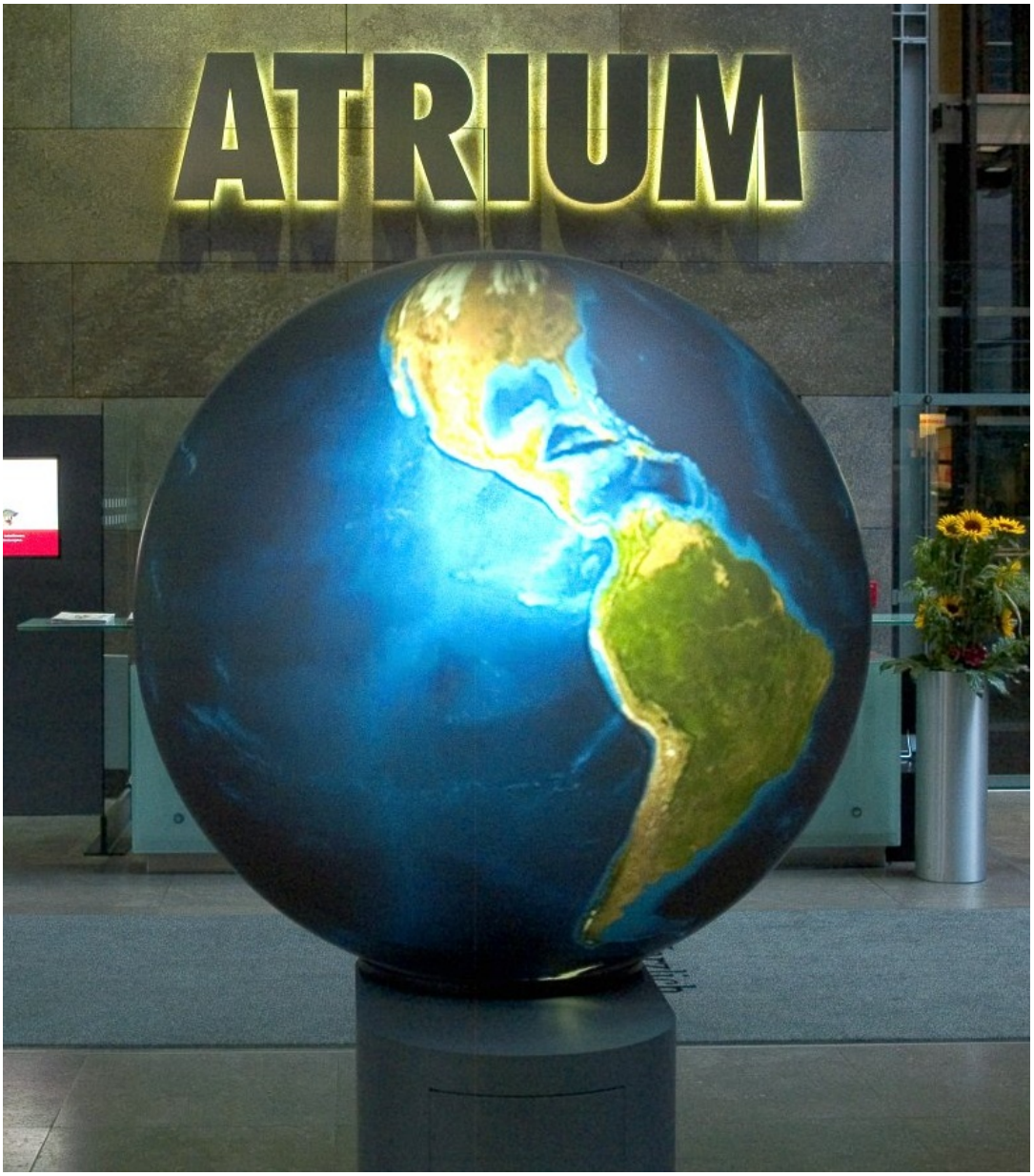

Abbildung 2.4: Der Omniglobe.

Kapitel 3

Technisches Umfeld

In diesem Kapitel werden die benötigten technischen Details vorgestellt. Der erste Abschnitt des Kapitels beschreibt das Projektionssystem. Es werden die technischen Details des Aufbaus sowie die Softwarekomponenten des Systems geschildert. Abschnitt 3.2 widmet sich dem Aufbau der Szene für die Darstellung mithilfe der Rendering-Software des Projektionssystems. Im letzten Abschnitt werden die grundlegenden Eigenschaften der eingesetzten standardisierten Technologien und Frameworks sowie ihre Zusammenhänge präsentiert.

3.1 Aufbau und Funktion des Projektionssystems

Im Inneren des Omniglobe arbeiten Hochleistungsprojektoren, etwa von JVC, Panasonic oder Projectiondesign, und patentierte optische Systeme im Zusammenspiel, die auf dem speziell beschichteten Kugeldisplay für eine perfekte Darstellung sorgen. Der Omniglobe existiert in drei Fassungen: 50 cm, 1,5 m und 3 m Durchmesser. Entsprechend der Größe wird der Omniglobe anders gebaut, da man z. B. für die 50 cm große Version bereits mit einem Projektor eine ausreichende Lichtstärke erhält.

Das während der Entwicklung benutzte Exemplar ist ein 1,5 m großer Omniglobe, der aus folgenden Teilkomponenten besteht:

- zwei Projektoren (Projectiondesign),
- Unterspiegel und ellipsenförmiger Oberspiegel,

- speziell beschichtetes Kugeldisplay,
- Shuttle-Form-Rechner (Quad Core CPU, Nvidia 8800GTX, 1Gb RAM),
- spezielle 3D-Rendering-Software,
- Wireless Access Point (Buffalo, DD-WRT Firmware),
- Stereolautsprecher,
- zwei Lüfter, Gehäuse, Master-/Slave-Steckdose.

Die Projektoren werfen das Bild auf die Unterspiegel, die das Licht nach oben durch einen Schacht reflektieren. Oben befindet sich der ellipsenförmige Oberspiegel, der dementsprechend die Oberfläche des speziell beschichteten Kugeldisplays belichtet. Eine schematische Veranschaulichung der oben erläuterten Funktionsprinzipien sieht man in der Abbildung 3.1.

Die Projektionssysteme mit mehr als einem Projektor bergen besondere Herausforderungen, wie Edge Blending und Farbkalibrierung. Der Kalibrierungsprozess betrifft dabei alle Ebenen des Projektionssystems von der Optik bis zu den Softwaremethoden, wobei Eigenschaften wie fokale Distanz, Alter der Lampen, Farbtemperatureinstellungen der Projektoren, die Marke sowie die Serie der Projektoren von Bedeutung sind.

Als Farbkalibrierung wird die uniforme Einstellung hinsichtlich Farbe und Helligkeit über alle Projektoren bezeichnet. Alle herkömmlichen Projektoren weisen unterschiedliche Farbräume auf, was selbst für Projektoren gleichen Typs gilt.

Edge Blending lautet der Termin für Methoden, die man anwendet, um den Übergang zwischen zwei Projektionen unsichtbar zu machen. Dabei unterscheidet man zwischen zwei Methoden:

Soft-Edge Blending Bei Soft-Edge Blending geht man davon aus, dass die Projektionen sich überlappen. Im Überlappungsbereich zeigen beide Projektoren denselben Bildinhalt, jedoch mit unterschiedlicher Helligkeit. Damit die Kante unsichtbar wird, wird die Helligkeit in den Überlappungsbereichen durch eine Maske gedämpft, sodass die Summe zweier Bilder den originalen Inhalt des Bildes ergibt.

Hard-Edge Blending Beim Hard-Edge Blending sind die Stoßkanten so exakt ausgerichtet, dass es keine Überlappung oder Spalte zwischen ihnen gibt. Auf diese Weise wurde z. B. das Blending-Problem bei HEyeWall gelöst, da Soft-Edge Blending nicht adäquat funktionierte.

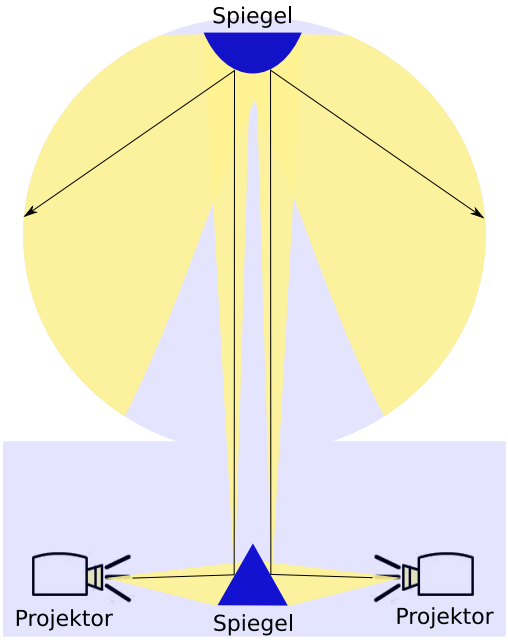

Abbildung 3.1: Schematische Darstellung des Omniglobe

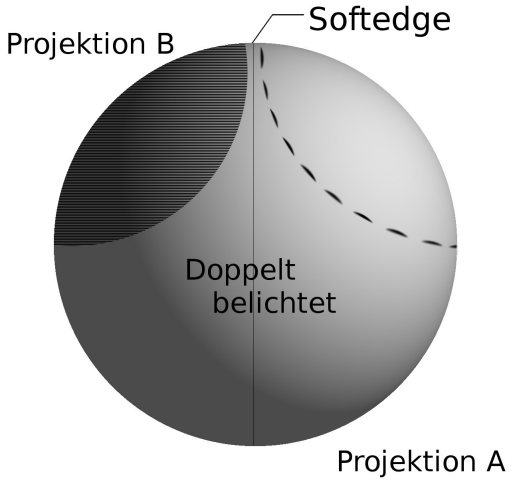

Abbildung 3.2: Überschneidung von Bildern beider Projektoren

Beim Omniglobe wird die Soft-Edge-Technik angewandt, weswegen auch die Kante zwischen zwei Halbprojektionen als Softedge bezeichnet wird. Die schematische Darstellung der Projektionen ist Bild 3.2 zu entnehmen. Beispielweise ist der ausmaskierte Bereich unten auf dem Bild 3.4 zu sehen.

3.1.1 Softwarekomponenten

Eine der wichtigsten Komponenten des Projektionssystems bildet die Rendering-Software. Für die Präsentation auf dem Omniglobe werden zweidimensionale Bilder verwendet, z. B. eine Karte der Erde, die sphärisch gemappt wird, sodass der Omniglobe einen virtuellen Globus veranschaulicht. Der Rendering-Software obliegt die Hauptaufgabe, die Ausgabebilder für die beiden Projektoren zu erstellen. Zunächst werden die Bereiche für den “linken” und den “rechten” Projektor ausgewählt. Wie oben in Bild 3.2 gezeigt, überschneiden sich die Projektionen von zwei Projektoren. Dies bedeutet, dass die Ausgabebilder für die Projektoren auch überschneidende Bereiche beinhalten müssen. In einem zweiten Schritt werden die ausgewählten Bereiche entsprechend dem Omniglobe-Oberspiegel transformiert. Ein anschauliches Beispiel der Transformationen, welche die Rendering-Software

durchführt, bieten die Bilder 3.3 und 3.4. Bild 3.3 präsentiert die sogenannte Grid-Textur, die für den Kalibrierungsprozess herangezogen wird. Die Textur soll einfach ein Gitternetz auf dem Omniglobe visualisieren, wie z. B. das Gradnetz der Erde.

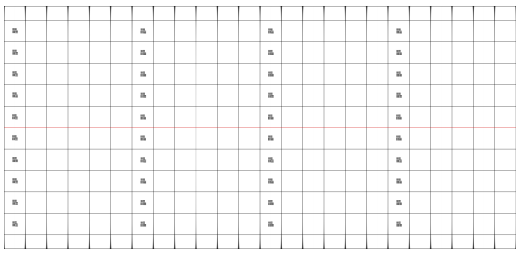

Abbildung 3.3: Eingangsbild

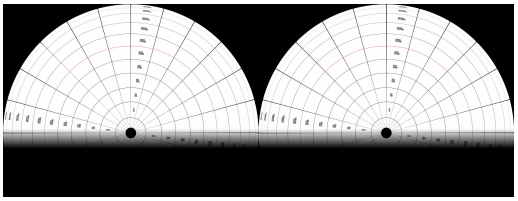

Abbildung 3.4: Die resultierenden Bilder für Projektoren

Die erste Version der Software hat für diese Zwecke Offline Renderer eingesetzt. Aus den Eingangsbildern oder Videos wurden mithilfe der oben kurz erwähnten Verfahren Flash-Videos zur Dual-Head-Ausgabe produziert. Um eine Szene auf dem Omniglobe darzustellen, sollte man das Video einfach abspielen.

Diese Vorgehensweise bietet einen Qualitätsvorteil im Vergleich zum Real-Time Verfahren, das im Rahmen dieser kompletten Arbeit eingesetzt wird. Dagegen bietet die Real-Time Lösung den für vorliegende Zwecke enormen Vorteil der Interaktivität. Im Rahmen dieser Diplomarbeit wurde eine Real-Time Rendering Engine eingesetzt. Mehr über die eingesetzte Software wird im Abschnitt 3.2 erklärt.

3.1.2 3D-Rendering-Engine

Für das Rendering der Ausgabe wird das DevDelight-Framework benutzt, das im Folgenden als Engine oder Rendering-Engine bezeichnet wird. Das DevDelight-Framework setzt auf modulare Architektur. Bei der Entwicklung wurden mehrere OpenSource-Projekte und Bibliotheken eingesetzt, wie z. B.:

Irrlicht Engine ist eine freie Grafikengine, die von einem kleinen Team um Nikolaus Gebhardt entwickelt wurde.

AngelScript ist eine C-förmige Skriptsprache für eine einfache Bindung über eine Schnittstelle mit C- und C++-Programmen.

Eingebaute HTTP-Server ermöglichen eine einfache Steuerung des Systems mithilfe von AJAX-basierten HTML-Seiten.

Sound- und Videosysteme Sound- und Videosysteme erlauben das Abspielen verschiedenster Sound- und Videoformate.

Beim Starten der Engine wird die Konfigurationsdatei *devdelight.xml* aus dem Unterordner *plugs* eingelesen. Die XML-Datei besteht aus zwei Abschnitten: EngineSection und SceneSection. Die SceneSection weist folgende Unterabschnitte auf, die jedes Untersystem der Engine einzeln konfigurieren:

beschreibt die Konfiguration des grafischen Untersystems. Es werden Einstellungen wie Vollbildmodus, Treiber (OpenGL, Directx 9 oder Directx9m für den Multimonitor-Modus), Größe der Render-To-Texture-Textur, Einschalten der Effekte, Schriftart, Abstand zwischen den Stereokameras für den Stereomodus, die Farbtiefe usw. vorgenommen.

Grafix beschreibt die Konfiguration des grafischen Untersystems. Es werden Einstellungen wie Vollbildmodus, Treiber (OpenGL, Directx 9 oder Directx9m für den Multimonitor-Modus), Größe der Render-To-Texture-Textur, Einschalten der Effekte, Schriftart, Abstand zwischen den Stereokameras für den Stereomodus, die Farbtiefe usw. vorgenommen.

Network umfasst die Konfiguration der zwei folgenden netzwerkbasierten Steuerungsmethoden des Omniglobe:

SyncNetwork wird für die zeitliche Synchronisierung mehrerer Engines in einem lokalen Netzwerk mittels Broadcasting benutzt.

ControlNetwork wurde für die Entwicklung des Autorentools eingeführt und erlaubt die Steuerung der Engine durch Aufrufe ferner Funktionen über eine Netzwerkverbindung.

Script kann das AngelScript-Untersystem eingeschalten und die Auszeit für jede einzelne Ausführung von Skripten definiert werden. Bei Überschreiten der Auszeit unterbricht die Engine die Ausführung des Skripts.

WebServer beschreibt die Konfiguration des eingebauten HTTP-Server. In diesem Abschnitt kann man z. B. den Port setzen.

VideoCapture erlaubt die Setzung einer maximalen Anzahl von Videostreams im System. Unter Videostreams versteht man die Ausgabe einer Webkamera sowie eingelesener Videodateien.

Manager und Audio sind zwei noch unentwickelte Abschnitte für die Konfiguration von Szenen und Audiountersystemen entsprechend dem Manager. Die Einstellungen dieser Untersysteme dienen der Konfiguration des Hauptsystems.

Input, Physic und AI bilden die Abschnitte für die Konfiguration der Untersysteme, die für die Spielentwicklung relevant sind. Dies sind entsprechend folgende Untersysteme: Eingabegeräte (wie z. B. Wii-Fernbedienung der Wii-Konsole), Physik-Engine[4] und Künstliche Intelligenz, z. B. für komplexe Bewegungen der computergesteuerten Figuren.

Jedes Untersystem kann man in den entsprechenden Abschnitten ein- oder ausschalten. Die Untersysteme Input, Physic und AI werden beispielsweise nicht benutzt.

Der Content für den Omniglobe wird in Szenen eingeteilt. Mehr Informationen über den Aufbau und die Aufteilung der Szenen finden sich in Abschnitt 3.2. An dieser Stelle ist allein die Tatsache relevant, dass jede Szene eine XML-Hauptkonfigurationsdatei besitzt, welche die gesamte Szene beschreibt. Im SceneSection-Abschnitt werden alle Szenen, die gerade im System abgelegt sind, einfach aufgelistet, indem man einen Unterabschnitt „Include“ benutzt. Der Abschnitt enthält den relativen Pfad zur XML-Hauptkonfigurationsdatei der Szene.

Die Engine läuft in Laufzeit durch die Liste der Szenen. Jede einzelne Szene wird für die Dauer der vorgegebenen Anzeigezeit präsentiert. Sämtliche Einzelheiten der Konfiguration, wie z. B. Anzeigezeit, geometrischer Aufbau, Interaktion mit dem Benutzer, werden für jede Szene einzeln durch die XMl-Hauptkonfigurationsdatei beschrieben.

Das Listing 3.1 zeigt ein gekürztes Beispiel der *devdelight.xml*-Konfigurationsdatei.

3.2 Aufbau der Szene

Im Rahmen dieser Arbeit werden unter dem Begriff “Szene” eine oder mehrere Szene-XML-Konfigurationsdateien, Skripten und alle dadurch referenzierten Dateien verstanden, wie Modelle und Texturen. Um eine Szene auf dem Omniglobe darzustellen, wird eine bestimmte Struktur der Szene verwendet.

[4] Dieses Teil der Frameworks dient des Simulation physikalischer Prozesse sowie der Berechnung objektimmanenter Eigenschaften (z. B. Impuls).

Listing 3.1: Beispiel der *devdelight.xml*-Konfigurationsdatei

```
<?xml version="1.0 "  encoding="ISO8859-1"?>
<DelightConfig>
    <EngineSection>
        <Manager enable="true"  multithreaded=" false "/>
        <Grafix
            enable="true"
            driver=" directx9 "
            fullscreen =" false "
            stereo =" false "
            width="800"
            height="600"
            depth="32"
            shadow="false"
            shader="true"
            eyesSeperation ="0.9"
            eyesAngle="0.00"
            textureOptimize=" false "
            engineFont=" ./ includes / fonts / lucida2 .png"
            caption="Developers  Delight  Game Framework"
            rttSize ="2048 2048"/>
        <Audio enable=" false "
                driver="not  used"
                recalcTime="2"/>
        <Network enable="true">
          <SyncNetwork enable=" false "
                        asServer=" false "
                        host2Connect2=""
                        port="9000"
                        timeOut="2"/>
          <ControlNetwork enable="true"
                          port="4242"
                          timeOut="2"
                          rootDir=" ./ "/>
        </Network>

        <Script  enable="true"  timeOut="10"/>
        <Webserver enable="true"  port="80"  requestsToLog="true"  uploadPath=" ./ "/>
        <VideoCapture enable="true"  maxStreamCount="100"/>
    </EngineSection>


    <SceneSection>
        <Include  file =" ./ Scenes/ authoring/ authoring .xml"/>
    </SceneSection>
</DelightConfig>
```

3.2.1 Struktur der Szenen für den Omniglobe

In Abschnitt 3.1.1 wurde die Erstellung von Ausgabebildern erläutert. Um dieses Verfahren mit der Rendering-Engine nachzuahmen, wird zunächst das erwünschte 2D-Bild der Szene erstellt. Mit dem erstellten Bild wird ein Model der Halbsphäre von "innen" texturiert. Das Modell wird von der Öffnungsseite mit einer sogenannten Render-To-Texture(RTT)-Kamera betrachtet. Die RTT-Kamera erzeugt eine Textur oder ein Bild davon, was die Kamera sieht.

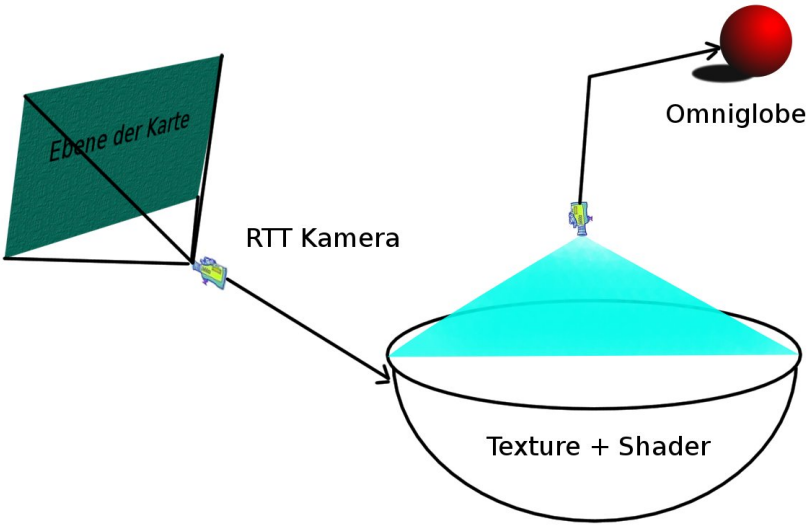

Abbildung 3.5: Aufbau einer Szene

Die Struktur der Szene ist Abbildung 3.5 zu entnehmen. Die Szene besteht aus folgenden Komponenten:

Die Ebene Karte Die "Karte" ist eine texturierte Ebene, die z. B. eine politische Karte der Erde veranschaulicht.

RTT-Kamera ist eine interne Render-To-Texture-Kamera. Diese Kamera liefert das 2D-Bild der Szene.

Halbsphere ist ein 3D-Modell, das zur richtigen Darstellung auf dem Omniglobe nötig ist. Auf dieses Modell wird das 2D-Bild der RTT-Kamera gemappt.

Erde Kamera ist eine Stereo-RTT-Kamera. Diese liefert zwei Bilder für die Projektoren.

Alles, was sich im Frustum[5] der RTT-Kamera befindet, wird in der daraus resultierenden Textur und entsprechend in der Halbsphäre und den Ausgangsbildern abgebildet. Auf diese Weise werden die Objekte, sogenannte Teilkomponenten der Szene, in die Szene eingeblendet.

Für die Erstellung der Struktur der Szene werden Konfigurationsdateien, Skripte und HLSL-Effekte verwendet. Die physikalische Aufteilung der Szene erfolgt mittels einer Ordnerstruktur. Jede Szene wird nach der Konvention in einem *plugs/Scenes/<Name der Szene>*-Unterordner gespeichert, der Szeneordner genannt wird. Die Szene besteht aus Scene-XML-Konfigurationsdateien, Skripten, Effekten und benutzten Ressourcen, wie 3D-Modelle, Texturen sowie Videos. Die Ressourcen befinden sich meist in demselben Szeneordner, können aber auch direkt referenziert werden.

3.2.2 Scene-XML-Konfigurationsdateien

Die XML-Konfigurationsdateien verknüpfen alle anderen Grundkomponenten. Durch diese Konfigurationsdateien werden die geometrischen Elemente, wie 3D-Modelle, eingelesen, an der richtigen Positionen platziert und mit den erwünschten Texturen sowie Effekten ausgestattet. In den speziellen XML-Abschnitten werden auch die Skripten und andere XML-Konfigurationsdateien berücksichtigt und korrekt initialisiert. Ein Beispiel der XML-Konfigurationsdatei ist Abschnitt A zu entnehmen.

Die Szene-XML-Konfigurationsdatei besteht aus folgenden Abschnitten:

Scene Der Abschnitt selbst bestimmt die Hauptparameter der Szene. Die wichtigsten sind Folgende:

- Einstellungen der Hauptkamera der Szene,
- Anzeigezeit für die Hauptschleife der Engine,
- Name der Szene,
- Hintergrundfarbe, Farbe des Schattens u. s. w.

Include in diesem Abschnitt können andere Szene-XML-Konfigurationsdateien in die Szene aufgenommen werden. Die Szene-Konfigurationsdatei, die man aufnehmen möchte, soll keine der Hauptparameter einstellen. Dieses Vorgehen ermöglicht eine

[5] Frustum ist ein "Pyramidenstumpf", der das Volumen visualisiert, welches vom Betrachter gesehen werden kann.

feine Aufteilung der geometrischen Konfigurationsdateien und Einstellungen, wie z. B. der Kamera.

ScriptInclude Damit können AngelScript-Skriptdateien in die Szene aufgenommen werden.

ScriptCallback definiert eine Rückruffunktion für die Engine-Ereignisse, wie Knopfdrücke, Mouse-Bewegungen, vordefinierte Timer-Ereignisse, Ereignisse beim Abspielen eines Videos und genericCallback für die WebServer-Interaktion mit dem Benutzer. Falls eine Rückruffunktion definiert wird, muss sich eine solche auch in den vorgegebenen Skriptdateien finden.

GeometricElement definiert einen geometrischen Bestandteil. Es existieren folgende Arten der Bestandteile: static, animated, text, terrain, plane, light, water, fire, beam, cloud, particle u. s. w. In diesem Abschnitt werden die geometrischen Parameter wie Position, Skalierung, Rotation etc. bestimmt; ferner wird die 3D-Modelldatei für dieses Element definiert. Unter anderem werden auch die Textur und die Effekte oder Shader für dieses geometrische Objekt angelegt.

LiveTexture definiert einen Videostream. Im Rahmen dieser Abhandlung wird dieses lediglich für das Inkludieren von Videodateien benutzt, ist aber auch zur Unterstützung von Webkameras und anderen Videogeräten entwickelt worden.

Effect definiert einen HLSL-Effekt. Im Abschnitt werden der Name und die Effektdatei gesetzt. An dieser Stelle wird der Effekt eingelesen und von einem HLSL-Kompiler vorkompiliert.

Die Szene-XML-Konfigurationsdateien beschreiben damit die statische Ausstattung der Szene und die Anbindung an Skripte, die für die dynamische Ausstattung der Szene zuständig sind.

3.2.3 Skripten

Zur Interaktion mit dem Benutzer sowie für komplexere Aktivitäten, wie z. B. Animation oder Timer-gesteuerte Ereignisse, kommt die Skriptsprache AngelScript zum Einsatz.

Die AngelScript-Sprache repräsentiert eine der C-formigen-Sprachen, die auch einiges aus der C++-Sprache übernommen hat. Diese Sprache lässt sich am besten anhand folgender Eigenschaften beschreiben:

Um die Interaktion mit dem Benutzer sowie die komplexere Aktivitäten, wie z.B. Animation oder Timer-gesteuerte Ereignisse, wird die Skriptsprache AngelScript eingesetzt.

Die AngelScript Sprache ist eine der C-formigen Sprachen, die auch einiges aus der C++ Sprache übernommen hat. Die Sprache ist durch folgende Eigenschaften am besten beschreibbar:

Statische Typen Im Vergleich zu vielen anderen Skriptsprachen werden die statischen Typen der C++-Sprache verwendet. Dazu werden noch die internen DevDelight-Typen, wie DS_Vec3d für die Vektorrechnung, registriert.

Objektorientierung Einige Eigenschaften der objektorientierten Programmierung werden ebenfalls unterstützt.

Objekt Handles Anstatt von Pointern werden die Objekt-Handles eingesetzt, die den Smart Pointern sehr ähnlich sind.

Sandboxing Die Skripten werden in einer Sandbox ausgeführt, das heißt, dass die Skripten nur zu den erlaubten Komponenten Zugang besitzen.

Kompilation im Laufzeit Die Skripten werden in Laufzeit vorkompiliert und danach in einer virtuellen Maschine ausgeführt.

Direkte Aufrufe Die Aufrufe zu registrierten Funktionen und Methoden werden ohne Einsatz von Proxyfunktionen realisiert.

Jede Szene enthält eine Domain und einen Namespace für Skripte, in welchen sich alle Skripten befinden, die die Szene lädt. Dies bedeutet ferner, dass die Namen der Funktionen, globalen Variablen sowie Klassen in jedem Skript unterschiedlich sein müssen. Die Szene-Konfigurationsdatei beschreibt die Ruckruffunktionen, die bei verschiedenen Ereignissen erfolgt. Es existieren folgende Typen der Ereignisse:

Listing 3.2: Beispiel des ScriptCallbacks

```
1 <Scene>
2   < ScriptInclude  path="./ preview. dscr"  section ="preview"/>
3   < ScriptCallback  function ="keyDownPreview" type="kd"/>
4 </Scene>
```

Dessen ungeachtet findet sich die “main”-Funktion, die bei jedem neuen Frame aufgerufen wird. Die Funktion wird für Initiierungszwecke herangezogen. Durch die Skripte werden die Aufrufe zu Funktionen der Rendering-Engine möglich. Im Listing 3.3 ist ein Beispiel

Beschreibung	Bezeichnung in der XML-Datei
Taste gedrückt	kd
Taste gelassen	ku
Mouse Move	mm
Mouse-Taste gedrückt	mbd
Mouse-Taste gelassen	mbu
MouseWheel wurde bewegt	mw
Abspielen der Audiodatei fängt an	playAs
Abspielen der Audiodatei stoppt	stopAs
Abspielen der Audiodatei pausiert	pauseAs
RayTracing wurde ausgeführt	ccr
Modelle kollidieren: Anfang	cmb
Modelle kollidieren: Mitte	cmd
Modelle kollidieren: Ende	cmf
HTTP-Requests	generic
Shader wird ausgeführt	sh
Effekt wird ausgeführt	ef
Timer-Ereigniss	timer

Tabelle 3.1: Typen der Ereignisse für Rückruffunktionen

der Init-Funktion vorgestellt. Tätigkeiten, wie z. B. Animation oder das Setzen der Texture-Rotation, sind nur durch direkte Aufrufe zur Rendering-Engine möglich.

3.2.4 Effekte

Unter dem Begriff *Effekt* sind im Rahmen dieser Arbeit High Level Shading Language (HLSL) Effekte gemeint. Die HLSL-Sprache ermöglicht die Programmierung der Hardware-Shader-Bausteine unter DirectX.

Shader oder Schattierer bilden Hardware- oder Softwaremodule, die bestimmte Rendering-Effekte bei der 3D-Computergrafik implementieren. Shader wurden ursprünglich ausschließlich für das Shading, mithin die Beleuchtungsberechnung, entwickelt, werden aber mittlerweile auch für andere Aufgaben verwendet. Historisch bedingt, existieren drei verschiedene Shader-Typen: Pixel-Shader, Vertex-Shader und Geometry-Shader. Dies ist der Tatsache geschuldet, dass die Grafikkarte verschiedene Hardwaremodule für die Bearbeitung von Pixel- und Vertexshader aufwies. Mit der Einführung von Shader Model 4.0 und somit auch Geometry Shader wurde auch die sogenannte Unified Shader Architecture auf den Markt gebracht.

Listing 3.3: Beispiel des Init-Skriptes

```
1
2  float  rotation_speed  = -0.2f;
3  bool  has_default_rotation_animator  = false;
4
5
6  void  init ()
7  {
8      // setting the position and aim for the edit cam
9      editCamPos = DS_Vec3f(1748.125366f, -76.874825f, -23.182610f);
10     editCamTar = DS_Vec3f(1752.250366f, -76.867081f, -23.181814f);
11
12     // setting the position and aim for preview cam
13     previewCamPos = DS_Vec3f(1.772825f, 7.438473f, -28.273623f);
14     previewCamTar = DS_Vec3f(1.746174f, 7.182462f, -27.307316f);
15
16     // Set the position of the RTT cam for editing mode
17     getGfxInstance().setRTTCamPosition("earth_kegel_cam0", editCamPos);
18     getGfxInstance().setRTTCamTarget("earth_kegel_cam0", editCamTar);
19
20     // add the default_rotation_animator to rotate the preview sphere or halfsphere
21     getGfxInstance().addAnimator("earth_kegel", "default_rotation_animator", DS_Vec3f
           (0.0f, 0.2f, 0.0f));
22     has_default_rotation_animator  = true;
23     // vertical flip of the texture of the karte object
24     getGfxInstance().setTextureRotation("karte", 0, DS_Vec3f(0.0f, -180.0f, 0.0f));
25     // vertical flip of the textureplane object
26     getGfxInstance().setTextureRotation("textureplane", 0, DS_Vec3f(0.0f, -180.0f, 0.0f)
           );
27
28     // setting and launching the "video" videostream
29     getGfxInstance().setTexture("videoplane", "video", 0, false, false);
30     getVideoInstance().play("video");
31
32     // vertical flip of the littleearth object
33     getGfxInstance().setTextureRotation("littleearth", 0, DS_Vec3f(0.0f, -180.0f, 0.0f))
           ;
34
35     // adding two lines
36     geoConnections.add(DS_Vec3f(-10, -400, -800), DS_Vec3f(-10, -400, 800), 4, DS_Vec4f
           (255, 100, 255, 255), "line", true);
37     geoConnections.add(DS_Vec3f(-10, 0, 0), DS_Vec3f(-10, 200, 0), 3.5, DS_Vec4f(255,
           255, 255, 255), "noname17", true);
38 }
```

Pixel-Shader dienen dazu, die zu rendernden Fragmente zu verändern, um beispielsweise eine realistischere Darstellung von Oberflächen- und Materialeigenschaften zu erreichen oder die Texturdarstellung zu verändern.

Vertex-Shader dienen dazu, die Geometrie einer Szene zu manipulieren, indem die Vertices (Eckpunkte dreidimensionaler Objekte) der Oberflächen verschoben werden. Allerdings kann der Shader nur bestehende Geometrien ändern.

Geometry-Shader kann neue primitive Geometrien aus bereits vorhandenen Primitiven (Punkte, Linien, Dreiecke) erzeugen und diese erneut in die Grafikpipeline einfügen.

Eine HLSL-Effektdatei beinhaltet die Definitionen und Quelltexte von Pixel-, Vertex- und möglicherweise auch Geometry-Shader, Konstanten, Texturen und sogenannte Texture-Sampler sowie den Technikenteil. Im Technikenteil werden die Einzelheiten des Renderingprozesses beschrieben, wie z. B. die Anzahl von Durchgängen (Passes). Durch die Benutzung von HLSL-Effekten kann man die Shader bequem vorkonfigurieren. Im Listing 3.4 ist der Pixel-Shader aus dem CityNoise-Effekt veranschaulicht. Das es sich hierbei lediglich um einen Ausschnitt handelt, ist es schwierig zu verstehen, woher die Werte stammen. Die Bezeichnungen “Noise” und “FlameSampler” stehen für eingeladene Texturen, wobei “Noise” eine dreidimensionale Perlin-Noise-Textur repräsentiert. “Timer”, “bubbleScale”, “bubbleRate” und “morphRate” sind float-Parameter. Die Parameter besitzen feste vordefinierte Werte, aber für jeden neuen Durchlauf (Frame) existiert auch die Möglichkeit, einen neuen Wert zu setzen. Die Werte und Texturen werden entweder durch die HLSL-Effektdatei definiert oder auch durch direkte Anweisungen im Effekt-Callback durch das Skript.

3.3 Benutzte Programmbibliotheken und Programmiertechniken

Wie viele andere technische Gebiete bildet Software Engineering ein sich sehr schnell entwickelndes Gebiet. Es entstehen neue Ideen und Ansätze, die auch zügig zum Einsatz kommen. Obwohl die „silberne Kugel“, die Frederick Brooks[6] gesucht hat, immer noch nicht gefunden wurde, sind bemerkenswerte Verbesserungen zu verzeichnen.

Die Entstehung neuer Entwicklungsparadigma, wie z. B. objektorientierte Programmierung oder aspektorientierte Programmierung, integrierte Entwicklungsumgebungen, ver-

[6] Vgl. [Brooks 1995]

Listing 3.4: Beispiel des Pixel-Shaders

```
float4 PS_Textured(vertexOutput IN): COLOR
{
    float2 pos = IN.texCoordDiffuse.xy;
    float time = (Timer * 10) % 400;

    // Bubble coords
    float3 coord;
    coord.xy = pos * bubbleScale;
    coord.z = bubbleRate * time;

    // Sample the noise for x and y
    float noiseX = tex3D(Noise, coord).r - 0.5;
    float noiseY = tex3D(Noise, coord + 0.5).r - 0.5;

    // Drop in the bubble noise
    coord.x = pos.x + noiseX * noiseScale;
    coord.y = pos.y + noiseY * noiseScale;
    coord.z = time * morphRate;

    // Sample the base noise
    float base = tex3D(Noise, coord).r;

    pos = (pos + 1) * 0.5;

    // cap the timer and cycle the animation
    if(time > 200) {
        time = 400 - time;
    }
    time *=3;

    float var = sin( time * pos.x * base * scale / 50 ) +
                cos( time * pos.y * base * scale / 50 );
    pos.x = (var+1) /3.5;
    pos.y = (var + 1.0) / 1.5;
    // another variant with sin curve applied too
    //pos.x = ((1.0 + sin(base))) / (time *0.25) ;
    //pos.y = ( ((1.0 + sin(base))) / (time *5.3) ) % 1.0 ;

    // sample the colour
    float3 colour = tex2D(FlameSampler, pos);

    // apply the calculated result
    float4 res = float4( colour, 1.0);

    // Assign a fire color from the base noise
    return res;

}
```

schiedene Entwicklungsprozesse, verstärkter Rückgriff auf erprobte Komponenten und Softwarebibliotheken, hat die Herstellung von Software deutlich erleichtert. Dennoch hat sich die Komplexität der entwickelten Software auch wesentlich erhöht.

C++ ist eine der am weitesten verbreiteten Sprachen, die auch ständig weiterentwickelt wird. In den vergangenen Jahren kam C++ sowohl in der Systemprogrammierung als auch in der Anwendungsprogrammierung zum Einsatz. Dabei gibt es die Tendenz, dass in der Systemprogrammierung C++ häufig den Platz der Assembler und Programmiersprache C einnimmt; aus der Domäne der Anwendungsprogrammierung wurde C++ mit dem Aufkommen der Sprachen Java und C# zum Teil zurückgedrängt. Anderseits sind viele Neuheiten solcher Sprachen, wie Java und C#, auch in die C++ umgesiedelt oder parallel entstanden.

Neue Bibliotheken, wie etwa Boost und die C++-Standardbibliothek, basieren auf der generischen Programmierung oder Metaprogrammierung, die eine der wichtigsten und weiterentwickelten Eigenschaften der C++-Sprache verkörpern. Die Bibliotheken bieten eine Vielfalt von Techniken, welche die Entwicklung der Software enorm erleichtern sowie die traditionellen Fallen der C++-Sprache besser umgehen, beispielsweise reguläre Ausdrücke, verschiedene intelligente Zeiger, ungeordnete assoziative Container, Hilfsmittel für die C++-Metaprogrammierung, Tupel sowie numerische und mathematische Bibliotheken.

In den weiteren Abschnitten werden die Programmbibliotheken sowie relevante Programmiertechniken beschrieben, die im Rahmen dieser Diplomarbeit herangezogen wurden.

3.3.1 Grafische Benutzeroberfläche

GIMP-Toolkit

Das GIMP-Toolkit oder GTK+ ist eine freie Komponentenbibliothek, mittels derer grafische Benutzeroberflächen (GUI) für Software erstellt werden können. Es ist eine in C geschriebene Bibliothek für eine grafische Benutzerschnittstelle, die ursprünglich als Alternative für Motif unter Linux entwickelt wurde, um somit dem Bildbearbeitungsprogramm GIMP eine Alternative für GUI bereitzustellen. GTK+ stellt eine Vielzahl an GUI-Elementen und weiteren Features zur Verfügung. Im Laufe der Zeit hat sich GTK+ zu einer der erfolgreichsten Widget-Bibliotheken entwickelt und ist auf X-Server basierenden Sys-

temen neben dem QT Toolkit auch eine der meist benutzten. GTK+ fungiert ferner als Basis der GNOME Desktop Environment.

GTK+ wurde in C-Sprache mit einem objektorientierten Ansatz implementiert, der die Anbindung an objektorientierte Sprachen, wie z. B. Python, Perl und C++, stark vereinfachte.

GTKmm

GTKmm repräsentiert eine portable und freie C++-Programmbibliothek, die eine Anbindung des in der Programmiersprache C geschriebenen GIMP-Toolkits an C++ erlaubt und als solche die bedeutendste extern entwickelte Anbindung von GTK+ bildet.

Die Bibliothek ist durch folgende Merkmale charakterisiert:

- Vererbung,
- Polymorphismus,
- typsicherer Signal-Slot-Mechanismus,
- STL-Einsatz,
- volle Unterstützung von UTF8,
- C++-Speicherverwaltung,
- Anwendung der C++-Namespaces,
- keine Makros,
- Plattformunabhängigkeit: Linux (gcc), FreeBSD (gcc), NetBSD (gcc), Solaris (gcc, Forte), Win32 (gcc, MSVC++, Net 2003), MacOS X (gcc) und andere,
- freie Software für Entwicklung quelloffener sowie kommerzieller Software,
- gute Dokumentation.

GTKmm besitzt eine Klassenhierarchie, welche die Entwicklung und Weiterentwicklung neuer Widgets stark vereinfacht. Durch die Typensicherheit von GTKmm werden einige Fehler im Quelltext bereits zur Kompilierzeit erkannt, was bei der GTK+-Bibliothek nicht der Fall ist. In GTK+ werden ausschließlich Zeiger verwendet, wovon man als C++-Programmierer Abstand nehmen und stattdessen vermehrt auf Membervariablen zurück-

greifen sollte. Membervariablen vereinfachen die Speicherverwaltung und lassen den direkten Einsatz der Zeiger zu.

GTK+ und GTKmm bauen auf diversen C-Bibliotheken auf, etwa Bibliotheken für den Umgang mit Datenstrukturen in C, portable Wrapper für betriebssystemabhängige Funktionen, und enthalten Schnittstellen zur Laufzeitfunktionalität, wie z. B. Ereignisschleifen, Threads, dynamisches Laden und ein Objektsystem, Layout- und Textrendering-Bibliothek mit dem Schwerpunkt auf Internationalisierung sowie eine Schnittstelle zu nativen Zeichenfunktionen eines Betriebssystems. Für einige Bibliotheken finden sich auch portable Backends.[7]

Glade und Libglade

Mithilfe von GTKmm und GTK+ kann man die Benutzeroberflächen ähnlich wie z. B. in Java vollständig im Source Code definieren. Als wesentlich komfortabler erweist sich die Verwendung eines "Interface Builders" wie Glade. Glade ist als unabhängiges Open-Source-Projekt zur intuitiven Erstellung von GTK+-Benutzeroberflächen entstanden. Mit Glade kann man ein GTK+-User-Interface schnell und einfach per Mausklick zusammenstellen. Es erzeugt keinen Programmcode für Ereignisse, sondern eine GLadeXML-Datei und auf Wunsch des Benutzers eine oder mehrere C-Dateien.

Libglade ist die Bibliothek aus der GTK+- und GTKmm-Sammlung, die das Laden der GladeXML-Dateien und Vererbung von Formen ermöglicht.

Das Listing 3.5 veranschaulicht die Anwendung der Libglade-Bibliothek. Zunächst muss die GladeXML-Datei eingelesen und geparst werden. Im Listing erfolgt das Einlesen und Parsen der *glade.xml*-Datei beispielsweise zwischen den Zeilen 29 und 38, indem die Methode *Gnome::Glade::Xml::create* aufgerufen wird. Die Methode liefert ein Objekt der *Gnome::Glade::Xml*-Klasse, welches die geparsten Daten der GladeXML-Datei enthält und die Bindung zwischen im Programm existierenden Klassen und in der XML-Datei vordefinierten Klassen erledigt. Objekte derselben Klasse werden ebenfalls an den Konstruktor der Klasse *ParameterFrame* übergeben, sodass der Konstruktor die internen Widgets aus der GladeXML-Datei, die diese Dateien beschreibt, initialisieren kann.

Nachdem die Instanz der *Gnome::Glade::Xml*-Klasse erzeugt wurde, gibt es zwei unter-

[7] Diverse Bibliotheken mit einem identischen Interface, welches man austauschen kann, werden in der Regel als Backends bezeichnet.

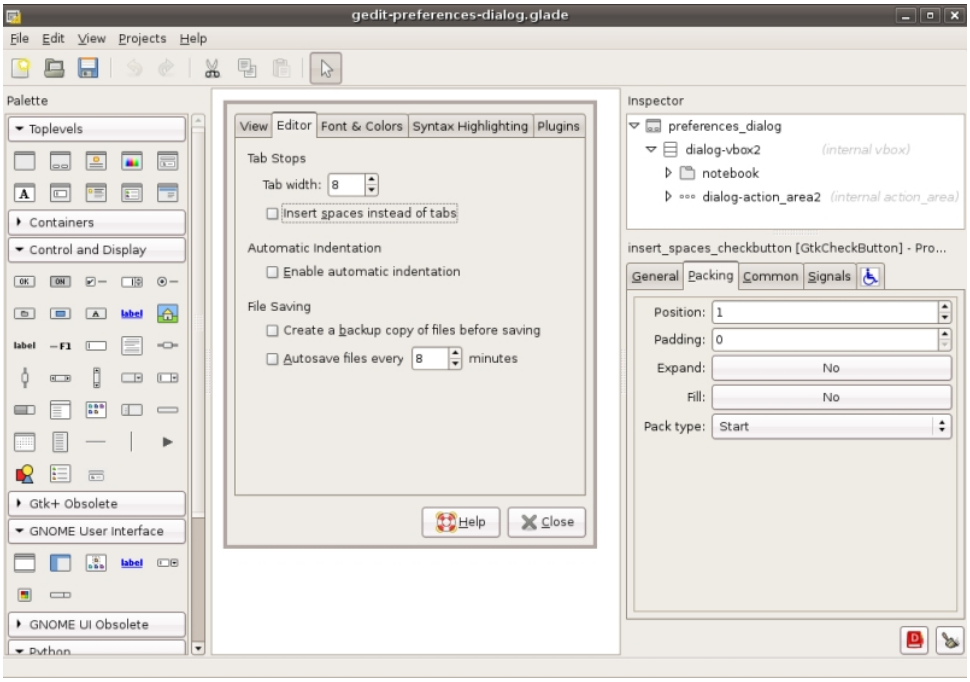

Abbildung 3.6: Glade “Interface Builder”

schiedliche Möglichkeiten, die Widgets aus der GladeXML-Datei und die Runtime Widgets zu binden.

Die erste Möglichkeit passt zur direkten Klasse-zu-Klasse-Bindung. In diesem Fall wird die *get_widget*-Methode der *Gnome::Glade::Xml*-Klasse benutzt. Die Methode sucht nach dem als ersten Parameter übergebenen Namen in der GladeXML-Datei und - falls das Widget mit diesem Namen vorhanden ist - erzeugt eine neue Instanz des Widgets; dabei setzt es die Parameter aus der GladeXML-Datei. Oben beschriebene Methode wird in den Zeilen 13 sowie 21, 23 und 26 benutzt. Die *refXml*-Instanz wird dem Konstruktor übergeben und danach für die Initialisierung einer Instanz der *Gtk::Viewport*-Klasse verwendet.

Anhand der zweiten Möglichkeit kann man die Klassen, die durch eine Vererbung entstanden sind, mit Parametern aus der GladeXML-Datei initialisieren. Die Zeilen 1-6 im Listing 3.5 deklarieren die Klasse *PlaneFrame*, die von der *Gtk::Frame*-Klasse vererbt wird. Um diese Klasse mit den Werten aus der GladeXML-Datei zu parametrisieren, wird die *get_widget_derived*-Methode der *Gnome::Glade::Xml*-Klasse benutzt (Zeile 34).

Listing 3.5: Einsatz der-Libglade Bibliothek.

```
class PlaneFrame : public Gtk::Frame, public GeometryObjectFrame
{
public:
    PlaneFrame( BaseObjectType* cobject , Glib :: RefPtr<Gnome::Glade::Xml>& refXml );
    ~PlaneFrame(void);
};

class ParameterFrame
{
    // Access to the .glade file
    Glib :: RefPtr<Gnome::Glade::Xml>&  refXml;

    Gtk::Viewport*                  viewport;
    PlaneFrame*                 plane;
public:
    ParameterFrame( MainWindow &mainwindow, Glib::RefPtr<Gnome::Glade::Xml> &refXml );
    ~ParameterFrame(void);
    // ...
};

ParameterFrame::ParameterFrame( MainWindow &mainwindow, Glib::RefPtr<Gnome::Glade::
    Xml>& refXml ) :
    mainwindow(mainwindow),
    refXml(refXml)
{
    // link the viewport widget
    refXml->get_widget("viewport", viewport);

    // load the elementary parameter frames
    try {
        // PLANE
        Glib :: RefPtr<Gnome::Glade::Xml> plane_refXml =
                Gnome::Glade::Xml:: create ("plane.glade", "frame");

        plane_refXml->get_widget_derived("frame", plane);
    } catch(Gnome::Glade::Xml::Error& xmlError) {
        // ...
        return;
    }

    // connect the outgoing signals of frames
    plane-> signal_failure_message () .connect( sigc :: mem_fun(mainwindow, &MainWindow::
        on_failure_message) );
    viewport->add(*plane);
    viewport->show_all();
}
```

3.3.2 Loggen

"Loggen" steht für die automatische Erstellung einer Logdatei (engl.: Log file) oder Protokolldatei, die alle oder bestimmte Aktionen von Prozessen auf einem Computersystem beinhaltet. Prinzipiell werden sämtliche Aktionen mitgeschrieben, für die ein späteres Audit erforderlich sind oder sein könnten.

EZLogger

Für die Entwicklung des Autorentools wurde der EZLogger gewählt. EZLogger ist eine C++-Bibliothek, die sich leicht integrieren lässt. Im Vergleich zu vielen anderen bekannten Loggen-Bibliotheken wie log4cxx, Boost Logging Library oder die Glog-Logging-Bibliothek von Google setzt sich diese Bibliothek nur aus Header-Dateien zusammen. Dies bedeutet, für die Integration mit dem Projekt muss man einfach die Header-Dateien in den Ordner des Projekts kopieren und die Hauptheader-Datei inkludieren. Es gibt keine vorkompilierte oder im Quelltext gelieferte Bibliothek, die Abhängigkeiten oder Versionsprobleme liefern kann. Die EZLogger-Bibliothek fungiert als gutes Beispiel einer Template-Bibliothek.

Trotz der leichten Ausführung bietet die Bibliothek umfangreiche Funktionalitäten:

- Stream-Operatoren und printf-Syntax,
- fünf in Laufzeit gesteuerte Logging-Niveaus,
- Loggen wird mithilfe von Makros durchgeführt, was die Ausgabe des Funktionsnamens, der Zeile und des Dateinamens ermöglicht,
- Verfolgung von Ein- und Austritt innerhalb der Funktionen,
- Debug-Modus-Makros für Nachrichten, die nicht ins Release kommen.

Weitere Informationen sowie Dokumentationen sind unter folgender Webadresse abrufbar: http://axter.com/ezlogger/.

Die Logdatei namens "ATool.log" wird im Arbeitsordner der Autorentools erstellt. Falls die Logdatei beim Starten der Anwendung existiert, wird sie überschrieben. Die Logdatei enthält Informationen über die wichtigsten Ereignisse, die innerhalb der Laufzeit der

Listing 3.6: Einsatz der EZLogger-Bibliothek.

```
// EZLOGGER SETTINGS
#define EZLOGGER_REPLACE_EXISTING_LOGFILE_
#define EZLOGGER_OUTPUT_FILENAME "ATool.log"

#include "ezlogger_headers.hpp"

bool test_function(int val)
{
    EZLOGGERVLFUNCTRACKER(axter::log_rarely);

    if (val == 5) {
        return true;
    }

    return false;
}

int main(int argc, char* argv[])
{
    int i = 123;
        std :: string somedata = "Hello World";

    // log verbosity
    axter :: ezlogger <>:: set_verbosity_level_tolerance (axter :: log_rarely);

    EZLOGGERSTREAM << "started" << std::endl;
    EZLOGGERVLSTREAM(axter::log_very_rarely) << "log_rarely verbosity level " << std ::
        endl;
    EZLOGGER_PRG_MAIN_ARG(argc, argv);


        //Simple usage with standard verbosity level
        EZLOGGERSTREAM << somedata << " " << i << std::endl;

        //Can use alternate stream
        EZLOGGERSTREAM2(std::cerr) << somedata << " next line " << i << std :: endl;

        // Verbosity level logging example
        EZLOGGERVLSTREAM(axter::log_often) << somedata << " " << i << std :: endl;

        //Complex extended data example
        EZLOGGERVLSTREAM(axter::levels(axter::log_often, axter :: warn, __FUNCSIG__ /*or
            GNU PRETTY_FUNCTION*/, "Xyz Facility")) << somedata << " " << i << std :: endl;

    test_function (5);
}
```

Anwendung geschehen sind, und hilft dem Entwickler dabei, mögliche Ursachen eines fehlerhaften Funktionieren der Software zu finden. EZLoggen unterstützt 5 Laufzeit-Logger-Ebenen: log_always, log_often, log_regulary, log_rarely, log_very_rarely. Bei der Entwicklung der Software wurden die Meldungen in vier Typen unterteilt:

- sehr wichtige Nachrichten: “log_always”,
- relevante Nachrichten: “log_regulary“,
- Debug-Nachrichten: “log_rarely“,
- Deep-Debug-Nachrichten: “log_very_rarely“.

Während der Entwicklung und für die erste Wartungsperiode wird der Deep-Debug-Modus als voreingestellter Modus belassen. Für die spätere Benutzung ist eine Umstellung auf den Relevante-Nachrichten-Modus geplant. Die Logdatei ist eine einfache Textdatei, die im folgenden Format geschrieben wird:

Name der Quelltextdatei (Zeile in der Quelltextdatei): [Klasse:Methode], [Stream der Logdatei] [Datum Zeit], Die Nachricht.

Dessen ungeachtet wird eine Logdatei der Rendering-Engine erstellt. Die Datei liegt im Unterordner */Engine/Log/* und wird auch bei jedem neuen Start überschrieben.

3.3.3 XML

Für das Parsen und die Erstellung der XML-Dateien kommt die Xerces-C++-Bibliothek zum Einsatz. Xerces-C++ ist eine schnelle, moderne, portable Bibliothek, die bereits im Rahmen der Entwicklung von DevDelight-Framework verwendet wurde. Die Bibliothek wurde als DOM-Parser und Generator benutzt.

3.3.4 Signal-Slot-Mechanismus

Signale und Slots sind ein Konzept der typsicheren Form von Rückruffunktionen (engl.: *Callback*). Sie realisieren einen ereignisgesteuerten Programmfluss bzw. eine ereignisgesteuerte Kommunikation zwischen Programmobjekten. Dieses Konzept ist bei den GUI-Bibliotheken sehr weit verbreitet.

Signale sind "Botschaften", die beim Eintreten eines Ereignisses abgegeben werden (emittiert). Ein Slot ist prinzipiell eine normale Funktion, die auf eine bestimmte Weise mit einem Signal verknüpft werden kann. Die Stärke dieses Systems besteht darin, dass die Slots und Signale erst durch die dynamische Verknüpfung die eigentliche Programmlogik bilden. Jedes Mal, wenn das Signal abgegeben wird, wird anschließend der verbundene Slot aufgerufen. Ein Signal kann auch mit mehreren Slots verbunden werden, sodass bei Eintreten eines Ereignisses mehrere Funktionen aufgerufen werden. Ebenso kann ein Slot mit mehreren Signalen verbunden werden, wodurch dieselbe Funktion bei Auftreten unterschiedlicher Ereignisse aufgerufen wird.

Es existieren verschiedene Programmbibliotheken, welche Signal-Slot-Mechanismen mittels generischer Programmierung und Funktoren realisieren. Die von GTKmm verwendete Bibliothek ist die SigC++, die ein typensicheres Callback-System in Standard C++ implementiert. Dies erlaubt dem Programmierer, Signale zu definieren und diese mit irgendeiner Rückruffunktion zu verbinden. Dabei können nicht nur Funktionen, sondern auch Methoden verwendet werden, selbst wenn diese statisch oder virtuell sind. Es beinhaltet auch Adapter, die es dem Programmierer ermöglichen, unterschiedliche Callbacks zu verbinden. Es besteht sogar die Möglichkeit, Standardparameter zu übergeben und damit eine Funktion mit mehreren Parametern an eine bestimmte Anforderung anzupassen.

3.4 Entwicklungstools

3.4.1 Versionsverwaltung

Für die Versionsverwaltung des Projekts wurde eine lokale Subversions-Repositary eingerichtet. Der Einsatz der Versionsverwaltungssoftware hat sich sogar für Projekte, die nur von einem Mitarbeiter getrieben werden, als deutlich sinnvolles Tool erwiesen. Die Versionsverwaltung dient der Kontrolle, Erstellung der Releases, Führung des Änderungsprotokolls und einfachen und sicheren Synchronisation von zwei Arbeitsstellen.

Listing 3.7: Beispiel der Signal-Slot-Technik

```
class CManager
{

public:
    void on_reload_scene(int i);

        typedef sigc :: signal<void, int> signal_update;
        signal_update signal_update () { return m_signal_update; }
protected:
        signal_update m_signal_update;
};

CManager::CManager(void)
{
    EZLOGGERVLFUNCTRACKER(axter::log_rarely);

    signal_update().connect( sigc :: mem_fun( *this, &CManager::on_reload_scene) );
    signal_update().emit(10);
}


void CManager::on_reload_scene(int i)
{
        EZLOGGERSTREAM << "*** Reloading the scene! ***" << std::endl;

        engine->reload_scene( Scene.get_name() );
        // etc ...
}
```

3.4.2 Dokumentation

Zur Erstellung der Dokumentation des entwickelten Systems wurde ein von Dimitri van Heesch entwickeltes Software-Dokumentationswerkzeug namens Doxygen herangezogen. Doxygen ist dem Software-Dokumentationswerkzeug Javadoc von Java sehr ähnlich. Beide Systeme dienen der automatischen Erzeugung von Dokumentationen. Durch spezielle Kommentare im Quelltext können Softwareentwickler Erläuterungen zu Programmelementen definieren, aus denen Doxygen eine übersichtliche Dokumentation erstellt. Ein Beispiel ist Abbildung 3.8 zu entnehmen.

Listing 3.8: Beispiel der Doxygen-Kommentare

```
1  /**
2   * \brief   Exemplarische Funktion
3   *
4   *          Diese Funktion gibt den uebergebenen Parameter
5   *          auf der Konsole aus.
6   *
7   * \param parameter    Auszugebender Parameter
8   * \return              Status-Code
9   *
10  */
11 int funktion(int parameter)
12 {
13      printf("Parameter: %d", parameter);
14
15      return 0;
16 }
```

Kapitel 4

Planung

Softwaretechnik (engl.: Software Engineering) widmet sich mit der kosteneffizienten Entwicklung qualitativ hochwertiger Software. Um dieses Ziel zu erreichen, wird das bereits seit Jahrtausenden bekannte Prinzip "Divide and Conquer" eingesetzt. Die Aufgabe wird in Teilgebiete oder Phasen gegliedert und nach und nach gelöst. Als erste und oftmals wichtigste Phase, die jedem erfolgreichen Projekt bevorsteht, wird die Planung erledigt. Laut Roger Pressman[8]:

> *"The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost and schedule."*

In der Planungsphase werden eindeutige und konkrete Definitionen der Anforderungen an die zu entwickelnde Software festgelegt sowie der Aufwand, die Projektgröße und strukturelle Unsicherheiten analysiert. Die Abschätzungen erfolgen vor einer systematischen und vollständigen Analyse und könnten somit eine bestimmte Ungewissheit beinhalten.

Dieses Kapitel widmet sich der Anforderungsdefinition mittels eines Pflichtenheftes in Abschnitt 4.2 sowie der Abschätzung der Kosten und des Aufwandes in Abschnitt 4.3.

4.1 Voranalyse und Anforderungen

Das Pflichtenheft ist ein Dokument, welches als Vorgabe das Lastenheft erfordert. Im Vergleich zu gewöhnlichen Industrie- und Geschäftsmethoden wurde für dieses Projekt kein

[8] Vgl. [Roger S. Pressman 2001]

Lastenheft erstellt, da der Autor dieser Arbeit an der Entstehung und Formulierung der Anforderungen teilgenommen hat. Das Pflichtenheft war dementsprechend das Ergebnis der Problemanalyse, die vom Entwickler (Auftraggeber) und Mitarbeitern der Fraunhofer Gesellschaft gemeinsam durchgeführt wurde. Das Lastenheft wurde damit obsolet.

Während der Voranalyse wurden folgende Aspekte der Entwicklung untersucht:

- Art der Software und Anbindung an DevDelight-3D-Engine oder eine andere 3D-Engine,
- Entwicklungsumgebung, Programmiersprache,
- Programmierbibliotheken für GUI, XML, Netzwerk u. s. w.

4.1.1 Design Varianten

Nachstehende Möglichkeiten wurden untersucht:

Plug-in für eine bereits existierende 3D-Grafiksoftware, wie z. B. 3ds Max oder Maya. *Vorteile:* Konsistente und bekannte Umgebung für den Benutzer. *Nachteile:* Kostenpflichtige Lizenz für Software wie Maya und 3ds Max, problematische Darstellung Echtzeit-Rendering-Szene in einer Offline-Rendering-Software, Komplexitäten beim Exportieren der Szene.

Aus diesen Überlegungen leitete sich die Anforderung ab, das DevDelight-Framework für das Rendering und die Darstellung der Szene beim Editieren zu nutzen. In einem ersten Schritt auf diesem Weg wurde folgender Ansatz untersucht:

Modul des DevDelight-Frameworks, das einen Editmodus implementiert. *Vorteile:* Keine Lizenzkosten, korrekte Darstellung der Szenen, direkter Zugang zu internen Klassen des Frameworks. *Nachteile:* Sehr starke Bindung beider Projekte, keine Möglichkeit zur Nutzung einer nativen GUI-Bibliothek des System.

Das DevDelight-Framework bietet keine Möglichkeit, native GUI-Bibliotheken, wie Window GDI, zu verwenden. Das DevDelight besitzt ein eigenes GUI-Framework, das auf dem GUI-Framework der Irrlicht-3D-Engine basiert. Eine andere Möglichkeit böte das CeGUI-Framework, das auch zur Anwendungen in 3D-Software entwickelt wurde. Diese GUI-Frameworks unterstützen lediglich einfachste Funktionen der nativen GUI-Framworks,

wie z. B. Knöpfe, ComboBoxes und RadioBoxes, und gehen dabei mit dem nativen GUI-Framework nicht konform. Die Benutzeroberflächen, die man mit z. B. CeGUI entwickelt, sind von Aussehen und Funktionsweise etwas anders. Letztendlich wandte sich der Entwickler des DevDelight-Frameworks gegen eine Zusammenführung beider Projekte.

Netzwerksteuerung des DevDelight-Frameworks wurde als letzte Möglichkeit vorgeschlagen. Für die GUI-Komponente wurde eine Netzwerkschnittstelle ControllNetwork-Handler entwickelt, die einen Zugriff auf die internen Funktionen der Engine realisiert. Die Schnittstelle implementiert eine Art Remote Procedure Call für das innere Interface der Engine. Die Schnittstelle wurde in Form einer dynamischen Bibliothek (DLL) realisiert. Der GUI-Teil bleibt dabei getrennt und kann beliebige Bibliotheken anwenden. *Vorteile:* keine extra Lizenzkosten fallen ein, korrekte Darstellung der Szenen. *Nachteile:* Anwendung der Remote-Procedure-Call-Schnittstelle.

Nach der Analyse wurden die ersten beiden Varianten abgelehnt. Die entwickelte Software sollte eine Anwendung repräsentieren, die keine extra Lizenzgebühren und großen Lernaufwand mit sich bringt. Die Anwendung sollte ferner eingeständig sein, sodass die Codebase der Rendering-Engine und Autorentools möglichst getrennt bleiben.

4.1.2 Entwicklungsumgebung und Programmiersprache

Das DevDelight-Framework wird in Visual C++ entwickelt. Obwohl die meisten Komponenten auch portabel sind, wurde die Portabilität der gesamten Anwendung als unnötig erachtet. Die Entwicklung der GUI-Komponenten unterscheidet sich deutlich von der Entwicklung der Low-Level-Framworks, wie etwa DevDelight-Framework. Aus diesem Grund wurde die mögliche Anwendung solcher Programmiersprachen wie C# und Java untersucht.

Es wurde festgestellt, dass die existierenden Klassen der DevDelight-Framework in Java nur sehr schwer zugänglich sind. Die Benutzung der Netzwerk-Bibliothek war lediglich durch eine Wrapper-Klasse möglich. Der Einsatz dieser Bibliothek verlangte dabei die Benutzung nativer Klassen der DevDelight-Engine, wie z. B. DS_Vec3f für dreidimensionale Vektoren. Anderseits ging man davon aus, dass der Anspruch an GUI nicht über die gewöhnlichen Grenzen hinausgeht. Diese Überlegungen führten zu der Entscheidung, die Entwicklung rein in Visual C++ und möglichst konform zur DevDelight-Engine zu halten.

4.2 Pflichtenheft

Das Pflichtenheft legt detailliert die Anforderungen an die zu fertigende Anwendung fest. Ziel ist es, aus den Anforderungen des Auftraggebers eine möglichst vollständige konsistente und eindeutige Produktdefinition zu erstellen, die als Grundlage für das zu erstellende Produkt dient. Das Pflichtenheft bildet ein verbindliches Dokument zwischen Auftraggeber und Softwareproduzent und ist Grundlage für den abzuschließenden Vertrag, nach dessen Unterzeichnung es nur im gegenseitigen Einverständnis geändert werden darf. Die Gliederung dieses Pflichtenheftes stammt von Prof. Balzert[9].

4.2.1 Zielbestimmungen

Ziel dieser Entwicklung ist ein Autorentool für das sphärische Projektionssystem Omniglobe. Das Autorentool soll die intuitive Erstellung von Szenen ermöglichen und die meistbenutzten Vorgänge automatisieren. Im Folgenden wird der Anwender des Autorentools als *Benutzer* bezeichnet.

Musskriterien

Der Benutzer bearbeitet die Szenen für den Omniglobe auf einem zweidimensionalen Arbeitsfeld. Auf dem Arbeitsfeld wird die Textur für das Mapping auf den Omniglobe erstellt.

- Der Benutzer kann neue Szenen anlegen, die an die spezifischen Anforderungen des Omniglobe angepasst sind.
- Die Szene kann als eine 2D-Ebene dargestellt werden.
- Der Benutzer kann Teilkomponenten (Objekte) in die Szene einfügen und löschen.
- Der Benutzer kann alle Parameter der Objekte bearbeiten.
- Die Änderungen sind sofort sichtbar.
- Der Benutzer kann sich die Szene als Realvorschau anzeigen lassen.

[9] Vgl. [Balzert 1998]

- Der Benutzer kann im Realvorschaumodus die präsentierte Sphäre beliebig bewegen.
- Der Benutzer kann die erstellte Szene zur Darstellung auf den Omniglobe exportieren lassen.

Die Szene besteht aus folgende, Teilkomponenten, die der Benutzer hinzufügen, bearbeiten und löschen kann:

- **3D-Modell** Der Benutzer kann beliebige 3D-Modelle, die von der DevDelight-Engine unterstützt sind, importieren und positionieren:
  - ◊ Name des Objekts,
  - ◊ die Modelldatei,
  - ◊ Position (im 3D-Raum der Szene),
  - ◊ Skalierung,
  - ◊ Rotation.
- **Ebene** Eine Ebene ist ein vordefiniertes quadratisches 3D-Modell, das der Darstellung von Videos, Texturen und Shader dient. Diese Teilkomponente erlaubt keine Änderung des Modells.
  - ◊ Name des Objekts,
  - ◊ Position (im 3D-Raum der Szene),
  - ◊ Skalierung,
  - ◊ Rotation.
- **Linie** Eine Linie verbindet zwei Punkte.
  - ◊ Name des Objekts,
  - ◊ Position des Anfangs (im 3D-Raum der Szene oder in Geokoordinaten),
  - ◊ Position des Ziels (im 3D-Raum der Szene und in Geokoordinaten),
  - ◊ Farbe,
  - ◊ Dicke.
- **Text** Text ist eine Teilkomponente, mithilfe derer man Textzeilen an beliebige Positionen in der Szene platzieren kann.
  - ◊ Name des Objekts,
  - ◊ Text,
  - ◊ Position (im 3D-Raum der Szene oder in Geokoordinaten),

  - ◊ Farbe.

- **Video** Video ist die Teilkomponente, die einen Videostream repräsentiert. Videos werden stets an geometrischen Teilkomponenten angewandt.
  - ◊ Name des Objekts,
  - ◊ Videodatei.
- **Textur** Texturen kommen an geometrischen Teilkomponenten zur Anwendung.
  - ◊ Name des Objekts,
  - ◊ Texturdatei,
  - ◊ Spiegelungen um die horizontale und vertikale Achse.
- **HLSL-Effekt** Diese Effekte werden aus einer Liste vorgefertigter Effekte ausgewählt.
  - ◊ Name des Objekts,
  - ◊ Name des Shaders

Wunschkriterien

- Der Benutzer kann die Szene als eine Projektdatei abspeichern und laden.
- Erweiterbarkeit um neue HSLS-Effekte ohne Rekompilation.
- Die entwickelten Szenen müssen möglichst durch komplexere Skripte und Shader erweiterbar sein.
- Der Benutzer kann die Szene in Echtzeit auf dem Omniglobe darstellen und bearbeiten.
- Der Benutzer kann die Szenen importieren lassen.

Abgrenzungskriterien

- Lediglich das Projektionssystem mit zwei Projektoren wird betrachtet.
- Die existierende 3D-Rendering-Engine ist als Blackbox zu betrachten und gestattet nur kleine Änderungen.

4.2.2 Einsatz

Die Anwendung ist primär für Anwender der Omniglobe-Projektionssysteme und Designstudios geeignet, die Szenen für den Omniglobe entwickeln wollen. Die Anwender benötigen keine Vorkenntnisse im 3D-Modelling.

Anwendungsbereiche

Potenzielle Anwendungsbereiche des Autorentools sind mit denen des Omniglobe identisch, mithin Messen, Museen, Präsentationen, Werbebranche sowie private Anwendung. Der Realvorschaumodus ermöglicht auch den Einsatz des Autorentools bei Entwicklern der Szenen, die kein Omniglobe besitzen.

Zielgruppen

Das System orientiert sich an den folgenden Benutzern:

- Designer,
- Grafiker,
- Entwickler der Effekte.

Betriebsbedingungen

Dieses System soll sich bezüglich der Betriebsbedingungen nicht wesentlich von anderen Content-Erstellungsanwendungen unterscheiden.

- Kein Dauerbetrieb,
- wartungsfrei,
- leicht anwendbar nach der Erstinstallation.

4.2.3 Produktumgebung

Der Abschnitt beschreibt die Software-, Hardware- und Orgware-Umgebung des Systems.

Software

- Betriebssystem: Windows XP SP3,
- DirectX 9.0c vom August 2008,
- DevDelight-Framework.

Hardware

Minimale Konfiguration:

- Pentium IV 3.0 Ghz,
- NVidia 6600GT Grafikkarte,
- RAM >= 1 GB,
- HDD >= 1 GB.

Empfohlene Konfiguration:

- Pentium IV 3.0 Ghz mit zwei Festplatten,
- NVidia 8800GTX Grafikkarte,
- RAM >= 2 GB,
- HDD >= 1 GB.

Orgware

- Das Autorentool muss für den Betrieb mit dem Omniglobe vorkonfiguriert werden:
- Netzwerkverbindung zum Omniglobe.

4.2.4 Produktfunktionen

Dieser Abschnitt beschreibt die einzelnen Funktionen des Systems.

Initiierung

Beim Starten des Systems

/F0010/ Initiierung der Anwendung. Zum erfolgreichen Starten des Systems sind folgende Abläufe erforderlich:

- ◊ Benutzeroberfläche wird initiiert und angezeigt.
- ◊ Der Renderer wird gestartet.
- ◊ Die Verbindung zwischen der Hauptanwendung und dem Renderer wird hergestellt.
- ◊ Die vordefinierte Szene wird geladen. Die Szene muss die Teilkomponente "karte" enthalten.

Addition, Entfernung und Bearbeitung der Teilkomponenten der Szene

Der Benutzer kann Teilkomponenten der Szene zur aktuellen Szene hinzufügen.

/F0020/ *Addition von einer 3D-Modell-Teilkomponente*

/F0030/ *Addition von einer Ebene Teilkomponente*

/F0040/ *Addition von einer Linie Teilkomponente*

/F0050/ *Addition von einer Text-Teilkomponente*

/F0060/ *Addition von einer Video-Teilkomponente*

/F0070/ *Addition von einer Textur-Teilkomponente*

/F0080/ *Addition von einer HLSL-Effekt-Teilkomponente*

Die Teilkomponenten bilden einen hierarchischen Baum. Die oben erwähnte Teilkomponente "karte" fungiert als Wurzel dieses Baums, alle anderen Teilkomponenten sind Kinder. Die Teilkomponenten können nur an geometrische Teilkomponenten (d. h. an ein Modell

oder eine Ebene) angehängt werden. Die Kinderanzahl ist vom Typ der Teilkomponente abhängig. Die neue Teilkomponente wird immer an die aktuell ausgewählte Teilkomponente angehängt. Die Regeln lauten wie folgt:

3D-Modell oder Ebene: unbegrenzte Anzahl von 3D-Modellen, Ebenen, Texten und Linien

3D-Modell oder Ebene: ein Video oder eine Textur

3D-Modell oder Ebene: ein HLSL-Effekt

alle anderen: keine Kinder

Der Benutzer kann Teilkomponenten aus der Szene herausnehmen. Bei der Entfernung der Vater-Teilkomponente werden die Kinder gleichermaßen entfernt. Die Teilkomponente "karte" kann nicht entfernt werden. Es wird stets die aktuell ausgewählte Teilkomponente entfernt.

/F0100/ *Entfernen einer 3D-Modell-Teilkomponente*

/F0110/ *Entfernen einer Ebene-Teilkomponente*

/F0120/ *Entfernen einer Linie-Teilkomponente*

/F0130/ *Entfernen einer Text-Teilkomponente*

/F0140/ *Entfernen einer Video-Teilkomponente*

/F0150/ *Entfernen einer Textur-Teilkomponente*

/F0160/ *Entfernen einer HLSL-Effekt-Teilkomponente*

Der Benutzer kann die Teilkomponenten der aktuellen Szene bearbeiten. Für die aktuell ausgewählte Teilkomponente wird ein Parameter Frame (siehe 4.2.7) angezeigt. Im Parameter Frame sind alle Eigenschaften der ausgewählten Teilkomponente veranschaulicht. Die Änderungen werden sofort im Renderer-Fenster oder auf dem Omniglobe abgebildet.

- **3D-Modell**

  /F0211/ *Änderung des Namens*

  /F0212/ *Änderung der Modeldatei*

  /F0213/ *Änderung der Position*

  /F0214/ *Änderung der Skalierung*

/F0215/ *Änderung der Rotation*

- **Ebene**

  /F0221/ *Änderung des Namens*

  /F0223/ *Änderung der Position*

  /F0224/ *Änderung der Skalierung*

  /F0225/ *Änderung der Rotation*

- **Linie**

  /F0231/ *Änderung des Namens*

  /F0232/ *Änderung der Anfangsposition*

  /F0233/ *Änderung der Zielposition*

  /F0234/ *Änderung der Farbe*

  /F0235/ *Änderung der Dicke*

- **Text**

  /F0241/ *Änderung des Namens*

  /F0242/ *Änderung des Textes*

  /F0243/ *Änderung der Position*

  /F0244/ *Änderung der Farbe*

- **Video**

  /F0251/ *Änderung des Namens*

  /F0252/ *Änderung der Videodatei*

- **Textur**

  /F0261/ *Änderung des Namens*

  /F0262/ *Änderung der Texturdatei*

  /F0263/ *Änderung der horizontalen Spiegelung*

  /F0264/ *Änderung der vertikalen Spiegelung*

- **HLSL Effekt**

  /F0271/ *Änderung des Namens/Änderung der Shader*

Szene

Die Szene hat einen Namen und eine Rotation. Unter Rotation der Szene wird die Rotation von der resultierenden Szene auf den Omniglobe verstanden.

/F0300/ *Änderung des Namens der Szene*

/F0305/ *Erstellung einer neuen Szene*

/F0310/ *Änderung der Rotation um die vertikale Achse*

/F0320/ *Speicherung der Szene in einer Projektdatei*

/F0330/ *Laden der Szene aus einer Projektdatei*

/F0340/ *Realvorschau:* Der Benutzer kann die Szene als Realvorschau ansehen.

Omniglobe

/F0350/ *Erstellung der Release Version der Szene:* Der Benutzer kann die aktuelle Szene für die Darstellung auf dem Omniglobe exportieren lassen.

/F0360/ *Herstellung der Verbindung mit dem Omniglobe.*

/F0370/ *Übertragung der Szene.*

4.2.5 Produktdaten

Die Anwendung erstellt die Szene-XML-Konfigurationsdateien und -Skriptdateien, für welche in den Abschnitten 3.2.2 und 3.2.3 entsprechend detaillierte Beschreibungen geliefert wurden.

Projektdatei

Die Anwendung speichert die Szene in einer Projektdatei. Die Projektdatei ist eine XML-Datei, die den derzeitigen Stand der Anwendung beschreibt.

Die XML-Datei enthält einen Wurzelknoten “project” und einen Baum aus “Object”-Knoten, welcher den Teilkomponentenbaum der aktuellen Szene visualisiert. Jeder Eintrag des Baums enthält zwei obligatorische Einträge “Name” und “Typ” sowie die optionalen Einträge in Abhängigkeit vom Objekttyp.

4.2.6 Produktleistungen

Die Leistungen definieren die Zeit- und Performance-Anforderungen an das System.

- Toleranz: Bei Fehler erzeugenden Eingaben muss der Benutzer über die Möglichkeit verfügen, eine Korrektur der Eingabedaten vorzunehmen, ohne Eingaben wiederholt eingeben zu müssen. Fehleingaben werden frühzeitig vermieden.

4.2.7 Benutzungsoberfläche

Es soll eine grafische Benutzeroberfläche entwickelt werden.

Hauptfenster

Das Hauptfenster setzt sich aus folgenden Komponenten zusammen:

- **Hauptmenü.**
- **Baumansicht der Teilkomponenten:** Diese stellt die Teilkomponenten der aktuellen Szene innerhalb einer Baumstruktur dar.
- **Parameter Frame:** Dieses veranschaulicht die Eigenschaften für die gerade ausgewählte Teilkomponente.
- **Werkzeugleiste:** Enthält alle Operationen, die dem Benutzer zugänglich sind.

Hauptmenü

Dieses enthält lediglich die wichtigsten Funktionen.

- File Menü
  - Open Project - **/F0330/**
  - Save Project - **/F0320/**
  - Save As Project - **/F0320/ + /F0300/**
  - Exit

- Scene Menü
  - New - **/F0305/ + /F0300/**
  - Build - **/F0340/**
  - Release Build - **/F0340/**
- Help Menu
  - About

Baumansicht der Teilkomponenten

Die Baumansicht visualisiert die Baumstruktur der Teilkomponenten für die aktuelle Szene und erlaubt die Auswahl einer der Teilkomponenten aus der Liste. Durch die Auswahl der Teilkomponente werden das Parameter Frame und die Werkzeugleiste gesteuert. Jeder Eintrag des Baums enthält zwei Spalten: den Namen und den Typ der Teilkomponenten.

Parameter Frames

Parameter Frames stellen die Eigenschaften für die gerade ausgewählte Teilkomponente dar. Für jeden Typ der Teilkomponente existiert ein Parameter Frame, der eine Änderung aller Funktionen für diesen Type (**/F02xx/**) mithilfe der GUI zulässt.

Es finden sich folgende Parameter Frames:

- Scene Frame (**/F0300/** und **/F0310/**)
- Model Frame (**/F0211/ - /F0215/**)
- Plane Frame (**/F0221/ - /F0225/**)
- Line Frame (**/F0231/ - /F0235/**)
- Text Frame (**/F0241/ - /F0244/**)
- Video Frame (**/F0251/ - /F0252/**)
- Texture Frame (**/F0261/ - /F0264/**)
- Shader Frame (**/F0271/**)

Werkzeugleiste

Diese enthält sämtliche Operationen, die dem Benutzer zu jedem Zeitpunkt zugänglich sind. Die Zugänglichkeit der Operationen ist von den gerade ausgewählten Teilkomponenten sowie deren Kindern abhängig.

- Verbindung mit dem Omniglobe oder Rendering-Engine herstellen,
- Release-Version der Szene bilden (**/F0340/**)
- Addition neuer Teilkomponenten (**/F0020/ - /F0080/**
- Entfernen der ausgewählten Teilkomponenten (**/F0100/ - /F0160/**

Dialoge

Im Folgenden wird die grobe Dialogstruktur einer fehlerfreien bzw. konfliktfreien Benutzung des Systems präsentiert. Fehlereingaben haben in der Regel einen Rücksprung auf die Ausgangsseite mit einer akkumulierten Fehlermeldung zur Folge.

- Eingabe des neuen Szenenamens.
- Scene-Exists-Replace-Dialog.
- Dateiimport für Texturen und Modelle.
- Speichern der Projektdatei.
- Laden der Projektdatei.

Renderer-Engine-Fenster

Das DevDelight-Framework wird in einem getrennten Prozess ausgeführt und besitzt ein eigenes Fenster, in dem die 3D-Szene dargestellt wird. Die Hauptanwendung steuert den Engine-Prozess, indem es z. B. in den Realvorschaumodus umschaltet.

Folgende Hotkeys sind im Engine-Fenster vorhanden:

e: (Edit) für Bearbeitungsansicht,

p: (Preview) für Realvorschaumodus,

der Linkspfeil rotiert die Sphäre im Realvorschaumodus nach links,

der Rechtspfeil rotiert die Sphäre im Realvorschaumodus nach rechts,

die Leertaste stoppt die Rotation im Vorschaumodus.

Im Engine-Fenster ist ferner die Auswahl der einzigen Teilkomponenten durch einen Mausklick möglich.

4.2.8 Qualitätszielbestimmungen

Zuverlässigkeit: Während des Betriebs muss die Konsistenz der Daten im System jederzeit gewährleistet sein. Daten dürfen lediglich im Sinne der funktionalen Anforderungen vom System verändert werden.

	sehr wichtig	**wichtig**	**weniger wichtig**	**unwichtig**
Robustheit		✓		
Zuverlässigkeit	✓			
Korrektheit	✓			
Benutzungsfreundlichkeit	✓			
Effizienz		✓		
Portierbarkeit				✓
Kompatibilität			✓	

Tabelle 4.1: Qualitätszielbestimmungen

Interaktionsanforderungen

Das System soll auf Anfragen und Bedienung jeglicher Art zeitnah, robust und deterministisch reagieren und sich zu keinem Zeitpunkt undefiniert oder instabil verhalten. Dies gilt insbesondere für das Speichern größerer Dateien.

4.2.9 Globale Testszenarien und Testfälle

Jede Produktfunktion wird anhand konkreter Testfälle getestet.

4.3 Abschätzungen

Laut L. Putnam und W. Myers[10] hängt die Umsetzbarkeit eines Projekts von folgenden vier Faktoren ab: Technologie, Finanzen, Zeit sowie Ressourcen. Für die Entwicklung des Autorentools war allein die technologische und zeitliche Umsetzbarkeit wichtig. Um die Analyse möglichst genau durchzuführen, wurde ein Prototyp des Systems erstellt. Der Prototyp bestand aus einem kleinen Ansatz der ControlNetworkHandler-Bibliothek. Die Anwendung hat einfach die Verbindung zur Rendering-Engine hergestellt, die Position der Kamera gesetzt und die Verbindung abgebrochen. Nach Beginn einer Analyse und eines Entwurfs des Systems erfolgte eine Untersuchung der Bibliotheken und deren Kompatibilität mit der Entwicklungsumgebung sowie dem DevDelight-Framework.

Anhand dieses Prototyps wurden die technologische Umsetzbarkeit des Projekts abgeschätzt. Zu diesem Zweck wurden zahlreiche Produktfunktionen aus dem Pflichtheft untersucht.

Außerdem wurden mithilfe des Prototyps die Kompatibilität und Funktionalität verschiedener Programmierbibliotheken getestet und ausgewählt. Die dabei ausgewählten Programmierbibliotheken wurden im Kapitel 3.3 beschrieben.

Anhand stattgefundener Voranalysen, existierender Erfahrungen und Kenntnisse wurden folgende Abschätzungen erstellt:

Während der Untersuchung wurden Existenz und Zugänglichkeit mehrerer Grundklassen des DevDelight-Frameworks berücksichtigt, wie z. B. 3D-Vektor Klasse DS_Vec3f. Gleichwohl ließen sich die technologische Komplexität und der zeitliche Aufwand einiger Produktfunktionen nicht einfach abschätzen.

Das Ergebnis der Abschätzung ist Tabelle 4.3 zu entnehmen. Der Stern in der Klammern innerhalb der Spalte “vor. zeitlicher Aufwand” markiert genau die Produktfunktionen, die sich zum Zeitpunk der Planung lediglich mit bestimmter Ungewissheit abschätzen lassen. Dies ist insbesondere Unsicherheitsfaktoren externer Abhängigkeiten geschuldet, wie z. B. der ControlNetworkHandler-Bibliothek des DevDelight-Frameworks. Aus der Tabelle geht hervor, dass das Projekt trotz seiner Größe eine hohe Anzahl der Funktionen enthält, die unüberschaubare technische Schwierigen bringen können. Aus diesen Gründen wurden die klassischen Lines-Of-Code-Abschätzungen außen vor gelassen. Basierend auf die-

[10] Vgl. [Putnam und Myers Nov]

sen Überlegungen, wurde ein vorläufiger Zeitplan entwickelt, nach welchem in den nächsten Monaten die Anwendung entwickelt wird.

Entwicklung	2.5 Monate
Tests, Evaluierung	2.5 Monate
Erstellung der Dokumentation	1 Monat

Produktfunktion	vor. zeitlicher Aufwand	Komplexität
/F0010/ Initiierung der Anwendung	2 Wochen	mittel
/F0020/ Addition des Modells	1 Tag	gering
/F02x1/ Änderung des Namens	1 Tag (*)	gering
/F0212/ Änderung der Modeldatei	1 Tag (*)	hoch
/F0213/ Änderung der Position	1 Stunde	sehr gering
/F0214/ Änderung der Skalierung	1 Stunde	sehr gering
/F0215/ Änderung der Rotation	1 Stunde	sehr gering
/F0030/ Addition der Ebene	1 Tag	gering
/F0223/ Änderung der Position	1 Stunde	sehr gering
/F0224/ Änderung der Skalierung	1 Stunde	sehr gering
/F0225/ Änderung der Rotation	1 Stunde	sehr gering
/F0040/ Addition der Linie	3 Tage (*)	mittel
/F0232/ Änderung der Anfangsposition	1 Stunde	sehr gering
/F0233/ Änderung der Zielposition	1 Stunde	sehr gering
/F0234/ Änderung der Farbe	1 Stunde	sehr gering
/F0235/ Änderung der Dicke	1 Stunde	sehr gering
/F0050/ Addition des Textes	1 Tag	mittel
/F0242/ Änderung des Textes	1 Stunde	sehr gering
/F0243/ Änderung der Position	1 Stunde	sehr gering
/F0244/ Änderung der Farbe	1 Stunde	sehr gering
/F0060/ Addition des Videos	2 Tage	mittel
/F0252/ Änderung der Videodatei	1 Tag	mittel
/F0070/ Addition der Textur	2 Tage	mittel
/F0262/ Änderung der Texturdatei	1 Tag	mittel
/F0263/,/F0264/ Änderung der Spiegelung	4 Stunden	gering
/F0080/,/F0271/ Addition des HLSL-Effektes	1 Woche (*)	hoch
/F0100/ - /F0160/ Entfernen von Teilkomp.	1 Woche (*)	mittel
/F0300/,/F0305/ Änderung des Namens	1 Tag (*)	mittel
/F0310/ Änderung der Rotation	1 Tag	mittel
/F0320/ Speicherung der Projektdatei	3 Tage (*)	mittel
/F0330/ Laden der Projektdatei	3 Tage (*)	mittel
/F0340/ Realvorschau	bis zu 1 Woche (*)	mittel
/F0350/ Erstellung der Release	2 Wochen (*)	sehr hoch
/F0360/ Verbindung mit Omniglobe	1 Tag (*)	mittel
/F0370/ Übertragung der Szene	1 Tag (*)	mittel

Tabelle 4.2: Abschätzung der Umsetzbarkeit

Kapitel 5

Implementierung

Aufbauend auf den Anforderungen des Pflichtenheftes, wird in diesem Kapitel das realisierte Softwaresystem beschrieben. Die Dokumentation der Implementierung wurde in vier Teile gegliedert: Beschreibung des Lösungsweges, Benutzeroberfläche, Dataklassen und Netzwerkschicht. Dabei werden für jede Ebene alle Angaben - von den Package-Diagrammen bis hin zu den Konfigurationsdetails - eigenständig veranschaulicht.

Vorgehensmodell

Die Entwicklung dieses Systems wurde auf der Basis des Rational-Unified-Process als objektorientiertes Vorgehensmodell für die Durchführung von Softwareprojekten unterstützt. Das Modell basiert auf der Unified Modeling Language und besteht aus folgenden Phasen:

- Geschäftsprozessmodellierung
- Anforderungsanalyse
- Analyse und Entwurf
- Implementierung
- Test
- Einsatz

  unterstützende Arbeitsschritte
- Konfigurations- und Änderungsmanagement

- Projektmanagement
- Entwicklungsumgebung

Da zur Realisierung des Systems lediglich eine Person zur Verfügung steht, wurden die Phasen im Einzelnen nicht ausführlich dokumentiert, sondern ausschließlich zur Selbstkontrolle benutzt.

Gesamtarchitektur

Zunächst wird die Gesamtarchitektur präsentiert. Die folgenden Abschnitte gehen ins Detail und erörtern die einzelnen Ebenen des Systems.

Das Autorentool besteht aus der Hauptanwendung und der Rendering-Engine, die durch die ControlNetworkHandler-Bibliothek gesteuert wird. Die Steuerung setzt sich aus RPC (Remove Procedure Call)-Aufrufen zu den registrierten Funktionen und Rückruffunktionen zusammen.

Die Hauptanwendung wurde in folgende drei Schichten unterteilt: Benutzeroberflächenschicht, Dataschicht und Netzwerkschicht. Außerhalb der Schichten stehen die Singleton-Klassen CManager, CConf und Logger-Klasse. Laut Eric Freeman:[11]

> *Singleton-Pattern sichert, dass es nur eine Instanz einer Klasse gibt, und bietet einen globalen Zugriffspunkt für diese Instanz.*

Genau aus diesen Gründen wurden die Manager, Konfigurationen und Logger-Klassen als Singleton-Klassen implementiert. Im Rahmen der Abhandlung wurde für diese Zwecke eine Klassentemplate-Singleton angewandt. Das Listing 5.1 illustriert den Einsatz dieser Klassentemplate. Anhand dieses Beispiels geht ferner hervor, dass für den einfacheren und klaren Zugriff auf die Instanz der Singleton-Klasse ein Makro verwendet wird.

Der Manager verkörpert eine Singleton-Instanz der CManager-Klasse, die alle Schichten der Hauptanwendung steuert und verbindet. Die globale Singleton-Instanz der CManager-Klasse dient als Zugangsknoten für alle Schichten. Die Schichten sind logisch voneinander getrennt und kommunizieren über den in Kapitel 3 beschriebenen Signal-Slot-Mechanismus oder durch indirekte Funktionsaufrufe über den Manager. Die Signale und Slots werden

[11] Vgl. [Freeman und Freeman 2006]

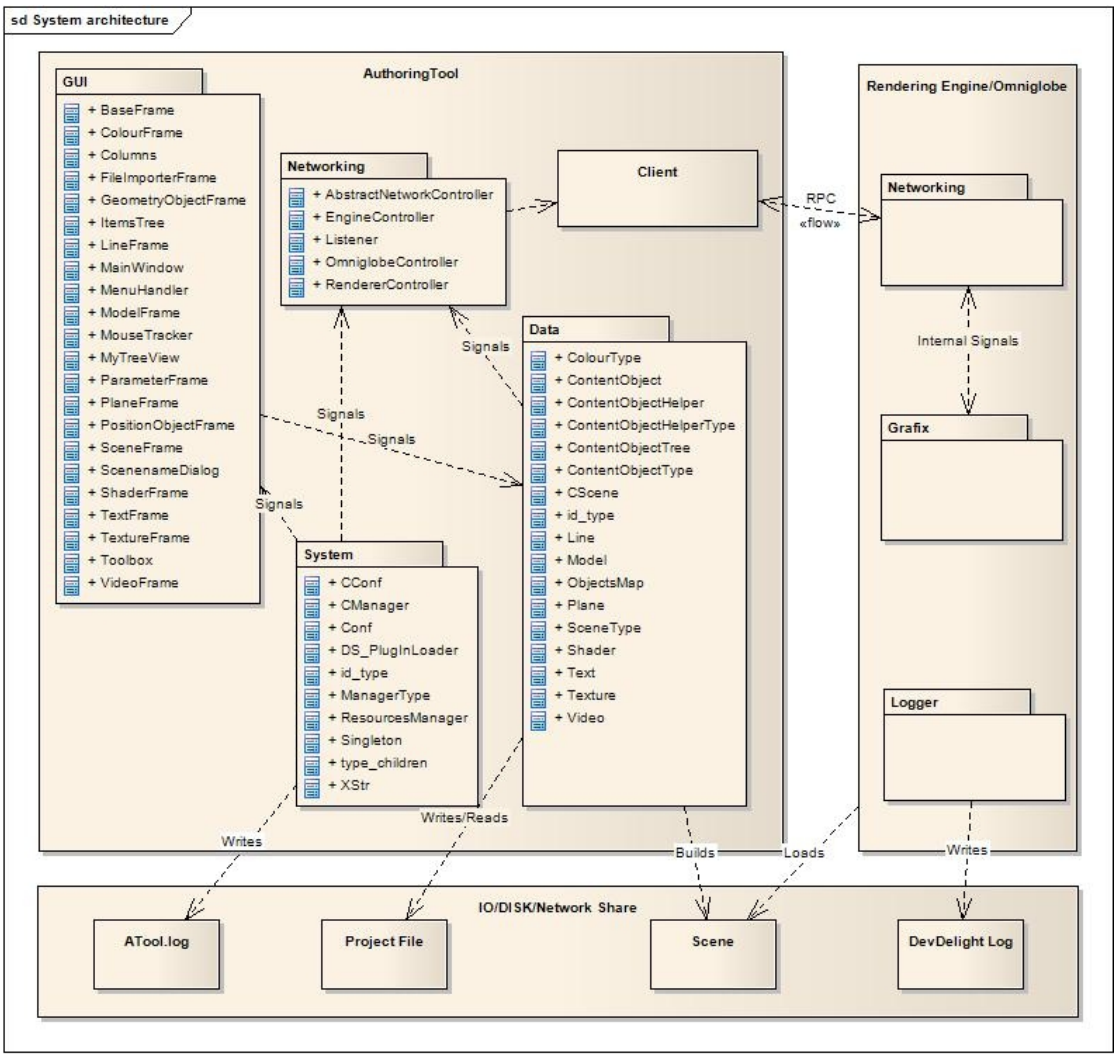

Abbildung 5.1: Systemarchitektur, logische Aufteilung

Listing 5.1: Beispiel der Template-Singleton-Technik

```
template <class T>
class Singleton {
public:
   static T& Instance() {
      static T _instance;
      return _instance;
   }
private:
   Singleton() {}                  // ctor hidden
   ~Singleton() {}                 // dtor hidden
   Singleton(Singleton const&);    // copy ctor hidden
   Singleton& operator=(Singleton const&);  // assign op hidden
};

class CManager
{
      // GUI Controller
      MainWindow *mainwindow;
      // ...
public:
      CManager(void);
      // ...
};
typedef Singleton<CManager> ManagerType;
#define Manager ManagerType::Instance()
```

meistens bei der Konstruktion der Klassen verbunden, wenn die Initiierung der Klasse beendet ist. Jede Schicht wird im folgenden Abschnitten einzeln ausführlich beschrieben.

5.1 Lösungsweg

Einzelne Funktionen des Autorentools wurden auf der Seite des Rendering-Frameworks implementiert. Für die Darstellung der Szene und der Realvorschau sowie der Linien wird eine speziell vorkonfigurierte “Authoring”-Szene herangezogen.

5.1.1 Realvorschau- und Bearbeitungsmodus

Die Hauptaufgabe der Rendering-Engine besteht in der Veranschaulichung des Arbeitsfeldes und dem Realvorschaumodus. Realvorschau- und Bearbeitungsmodus werden hauptsächlich auf der Seite der Rendering-Engine erledigt. Das Bearbeitungsfeld ist nichts ande-

res als die “Ebene Karte”, die texturierte Ebene, die in alten Szenen des Omniglobe bereits vorhanden ist. Um die Darstellung für den Bearbeitungsmodus zu ermöglichen, wurde eine neue “edit”-Kamera eingeführt, welche die identischen Parameter wie die RTT-Kamera besitzt. Wenn diese aktiviert wird, zeigt das Renderer-Fenster genau das Arbeitsfeld.

Die Realvorschau wird auf ähnliche Art und Weise realisiert. Die Entstehung des Ausgangbildes wurde in Kapitel 3 folgendermaßen geschildert:

1. Die Eingangsteilkomponenten und Bilder bauen eine “Szene” im dreidimensionalen Raum auf.

2. Eine RTT-Kamera (und die “edit”-Kamera) bildet daraus eine RTT-Textur.

3. Die RTT-Textur wird auf die innere Seite einer Halbsphäre gemappt.

4. Auf die Halbsphäre wird eine spezielle Stereo-RTT-Kamera gezielt, die zwei Ausgabebilder für die Projektoren liefert.

Für die Erstellung der Realvorschau wird in einem dritten Schritt die Halbsphäre durch eine skalierte Sphäre ersetzt. Da die RTT-Textur immer noch auf die innere Seite der Kugel gemappt wird, muss man die Sphäre mit den Werten [-1.0, 1.0, -1.0] skalieren. Im Preview-Modus wird die Rotation der Sphäre durch einen Animator gesteuert. Eine “preview”-Kamera ist aus einer Distanz auf die Sphäre gerichtet und liefert das Bild für den Realvorschaumodus. Die Ausgabebilder beider Moden sind auf den Bildern 5.2 und 5.3 entsprechend visualisiert.

Abbildung 5.2: Bearbeitungsmodus

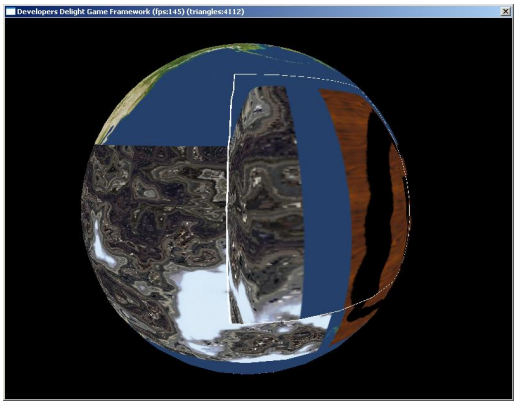

Abbildung 5.3: Realvorschaumodus

5.1.2 Umstrukturierung der Szene

Noch im Rahmen der Analyse der Anforderungen wurde folgende technische Herausforderung des entwickelten Systems erfasst. Sehr viele Produktfunktionen des Systems sind durch die Aufrufe zum Rendering-Engine nicht realisierbar. Als noch schlimmer erwies sich die Tatsache, dass diese nicht realisierbaren Produktfunktionen auch durch innere Methoden des DevDelight-Frameworks in den meisten Fällen nicht verwirklicht werden konnten. Tabelle 5.1.2 gibt eine Auflistung von Funktionen und deren Realisierbarkeit wieder. Wichtig dabei ist die letzte "Build-Reload"-Spalte. Ein Haken in dieser Spalte bedeutet, dass die erwünschte Änderung nicht ohne "build-reload"-Zyklus möglich ist. Die mittleren zwei Spalten beschreiben, welche Änderungen in der Szene gebracht werden, um die erwünschte Operation durchzuführen. Das Ergebnis ist der Abbildung 5.4 zu entnehmen. Die Abbildung illustriert alle Aufrufe zur build-Methode der CScene-Klasse.

Aufgabe	**Skript**	**XML**	**Build-Reload**
Ebene/Modell			
Addition/Löschen		✓	✓
Änderungen der Parameter		✓	
Laden des Modells		✓	✓
Textur			
Addition/Löschen		✓	
Spiegelung um die Achse	✓		✓
Transparenz		✓	✓
Video			
Addition/Löschen	✓	✓	✓
Stoppen/Abspielen	✓		
Shader			
Addition/Löschen	✓	✓	✓
Text			
Addition/Löschen	✓	✓	✓
Änderung der Parameter		✓	✓
Addition des Textes	✓		✓
Linie			
Addition/Löschen	✓		✓
Änderung der Parameter	✓		✓

Tabelle 5.1: Vergliech der Produktfunktionen

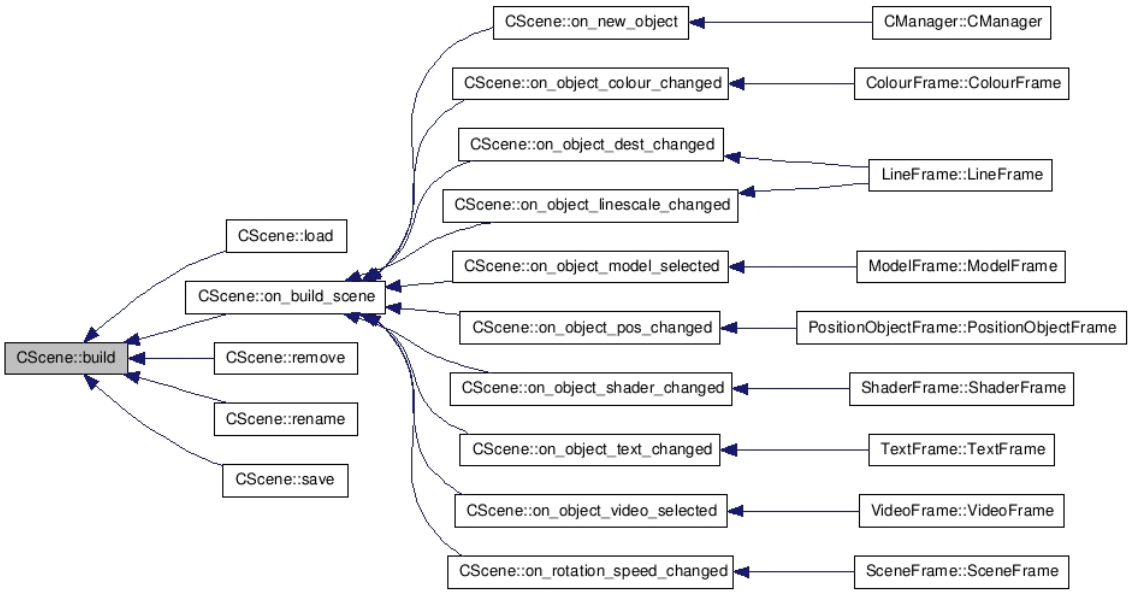

Abbildung 5.4: Aufrufe zur build-Methode

Die einzige Lösung bestünde zum einen darin, das DevDelight-Framework zu ändern, was offensichtlich zu einem Reengineering der meisten Komponenten des System führte, oder zum anderen darin, die Produktfunktionen durch einen “build-reload”-Zyklus zu realisieren.

Unter dem “build-reload”-Zyklus wird die Erstellung und das Laden einer neuen Szene verstanden, was die Erzeugung von XML-Konfigurationsdateien, Skripten, Effekten usw. bedeutet. Der “build-reload“-Zyklus, der zu den wichtigsten Ansätzen des Autorentools geworden ist, wird in Abschnitt 5.4.1 genauer beschrieben.

Aus diesem Ansatz resultierte indes ein Problem mit der Erweiterbarkeit der Szene. Die automatisierte Erstellung der Szene auf diese Art bedeutete, dass die Szene immer wieder neu überschrieben wird, womit alle nicht automatisierten Änderungen des Benutzers verloren gehen würden. Eine Lösung für dieses Problem wurde durch das Umstrukturieren der Szene gefunden.

Das Problem der originalen Struktur der Szene, die in Kapitel 3 erörtert wurde, bestand darin, dass lediglich eine Rückruffunktion jeder Art pro Szene existierte. Um dieses Problem zu umgehen, wurde der Callback-Mechanismus erweitert, sodass jede Szene-XML-Konfigurationsdatei mehrere Rückruffunktionen für dieselben Ereignisse definieren dürfte. Die mehrfach definierten Rückruffunktionen werden einfach eine nach der anderen aufgerufen. Ferner wurde ein Init-Callback-Ereignis eingeführt. Die Init-Funktion wird

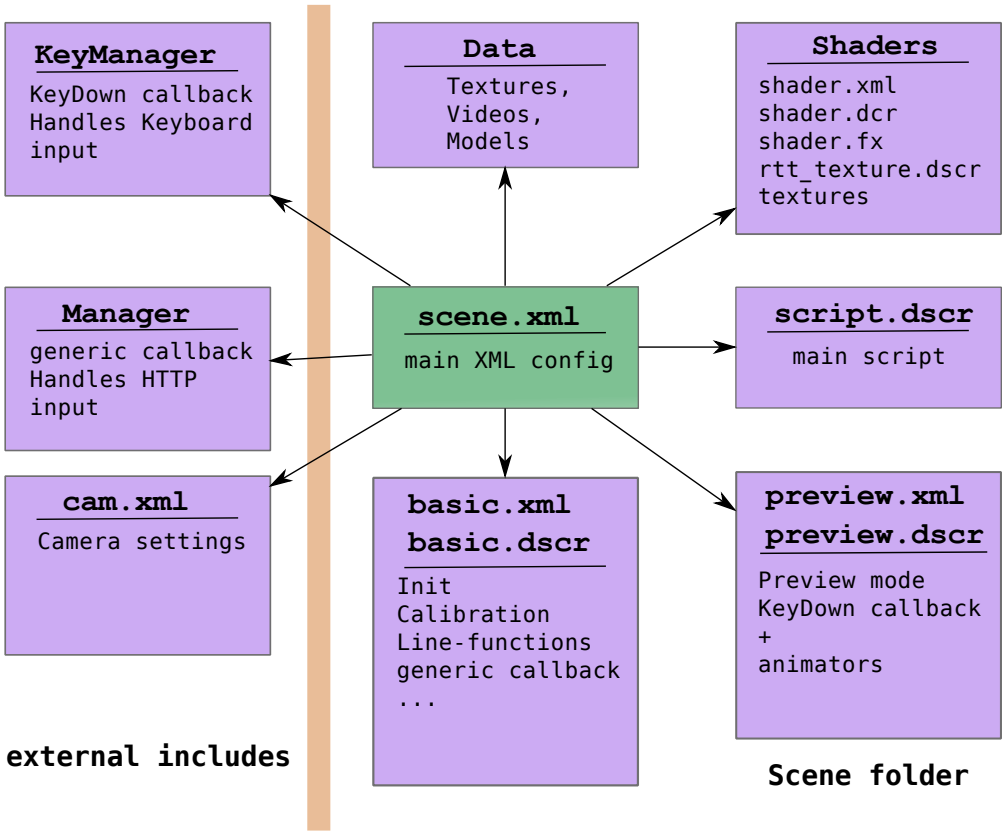

Abbildung 5.5: Struktur der Szene

lediglich einmal beim Laden der Szene betätigt. Damit wurde die Struktur der Szene deutlich modularisiert. Die Szene wurde in folgenden Komponenten geteilt:

- Grundfunktionalität: basicxml, basic.dscr,
- Realvorschau: preview.xml, preview.dscr,
- Hauptkomponente (wird vom Autorentool erstellt): <scenename>.xml, script.dscr.

Die Aufrufe zu Systemskripten, wie z. B. OmniglobeManager oder KeyDownManager, die früher stets in Callback-Funktionen des Hauptskriptes zu finden waren, wurden auf diese Weise einzeln in ein Modul aus XML-Konfigurationsdatei und einer Skriptdatei gepackt. Damit wurden die Struktur der Szene und die Skriptdatei deutlich vereinfacht. Die neue Struktur der Szene ist Abbildung 5.5 zu entnehmen.

Außerdem wurde die Pfadkonvention der Szene geändert. Um den Transport der Szene zu erleichtern, wurde entschieden, dass sich alle Komponenten der Szene in einem Szeneordner befinden müssen. Das bedeutet, dass z. B. die Modelle, Texturen und Videos stets im

Ordner der Szene verortet sind. Nur die wichtigsten Skripte und Konfigurationsdateien, wie z. B. Kameraeinstellungen (*cam.xml*), dürfen außerhalb des Szeneordners liegen. In diesem Fall müssen diese Komponenten an vordefinierten Stellen im *plugs*
Includes
...-Ordner platziert werden. Will der Benutzer etwa eine Textur anwenden, wird diese Textur automatisch in den Szeneordner importiert.

5.1.3 Implementierung der HLSL Effekte

Hinsichtlich der Unterstützung der HLSL-Effekte wurden Änderungen vorgenommen:

- Unterstützung der SAS („Standard Annotations and Semantics“),
- Auslagerung der Effekte in Module aus folgenden Bestandteilen:
  - HLSL-Effektdatei,
  - XML-Konfigurationsdatei,
  - Skript mit Rückruffunktion für Initiierung und Ausführung des Shaders,
  - alle Ressourcen für den Shader sowie Texturen, Noise-Texturen.

Für die Ausführung jeden Shaders oder Effekts werden bestimmte Parameter an den Grafikprozessor (GPU) übergeben. Dies sind beispielweise Texturen, Wert des Timers, Einstellungen der Kamera usw. Die Unterstützung von SAS kann größtenteils automatisch erfolgen. So etwa werden die Matrizen, welche die Kameraeinstellungen beschreiben, automatisch an jeden Shader übergeben, der eine Matrix mit dem vorgegebenen Name definiert.

Die Auslagerung der Shader-Initiierung und -Ausführung in eine getrennte Datei hat die Erzeugung der Szene deutlich erleichtert. Alle Effekte wurden umgeschrieben und in einem Shader-Ordner im Arbeitsordner des Autorentools abgelegt. Jeder Shader befindet sich in einem eigenen Unterordner, der denselben Titel trägt wie der Shader.

Auf diese Weise wurde die Prozedur der Inklusion von einem Shader auf das Klicken auf dem Namen des Shaders vereinfacht. Bei jedem Start der Anwendung wird der Shader-Ordner neu durchsucht. Wenn der Benutzer einen Shader anwenden will, erhält er eine Liste zugänglicher Shader. Hinsichtlich des Einsatzes der Shader wird Folgendes unternommen:

1. Der Unterordner des Shaders wird in den Szeneordner importiert.
2. Die XML-Konfigurationsdatei des Shaders wird in die Inkludes-Liste der Szene eingefügt.
3. Die Teilkomponente, an die der Shader angewandt wird, bekommt einen ”effect“-Eintrag.
4. Die Szene wird neu erzeugt und geladen.

Eine besondere Komplikation gibt es mit den RTT-Shadern oder Shadern, die an die Halbsphäre angewandt werden und die RTT-Textur als Eingangstextur nehmen. Diese Shader wurden als Spezialfall eingebaut und werden als RTT-Shader bezeichnet. Es darf nur einen RTT-Shader im System geben, der stets nur an “karte”-Teilkomponenten angewandt wird. Die “karte”-Teilkomponente darf dabei noch einen anderen als einen RTT-Shader anwenden. Die RTT-Shader werden von den normalen Shadern durch eine rtt_texturing.dscr Skriptdatei im Shader-Unterordner unterteilt, die die Übergabe der RTT-Textur von der RTT-Kamera der Szene steuert.

5.2 Netzwerkschicht

Die Netzwerkschicht des Autorentools wurde für flexible und transparente Kommunikation mit der Rendering-Engine entwickelt. Dafür wird die ControlNetworkHandler-Bibliothek des DevDelight-Frameworks verwendet. Ein Ausschnitt aus der gesamten Systemarchitektur - dargestellt in Abbildung 5.6 - illustriert die Bestandteile der Netzwerkschicht.

Die Netzwerkschicht wurde zu Beginn für den gleichzeitigen Ansatz des Omniglobe und der Rendering-Engine projektiert. Während der Entwicklung des Systems wurde die technische Schwierigkeit dieses Vorgehens festgestellt. Die ControlNetworkHandler-Bibliothek erlaubt nur eine Verbindung per .DLL, da die runtime-Daten der Verbindung im Datenbereich der DLL gespeichert sind. Dies ist mit dem Design des DevDelight-Frameworks verbunden und lässt sich allein durch ein komplexeres Laden der zweiten DLL-Datei lösen. Aus diesem Grund wurde die Arbeit in dieser Richtung eingestellt, aber die Klassen, wie z. B. OmniglobeController, wurden an der Stelle belassen.

Die Netzwerkschicht erledigt folgende Aufgaben:

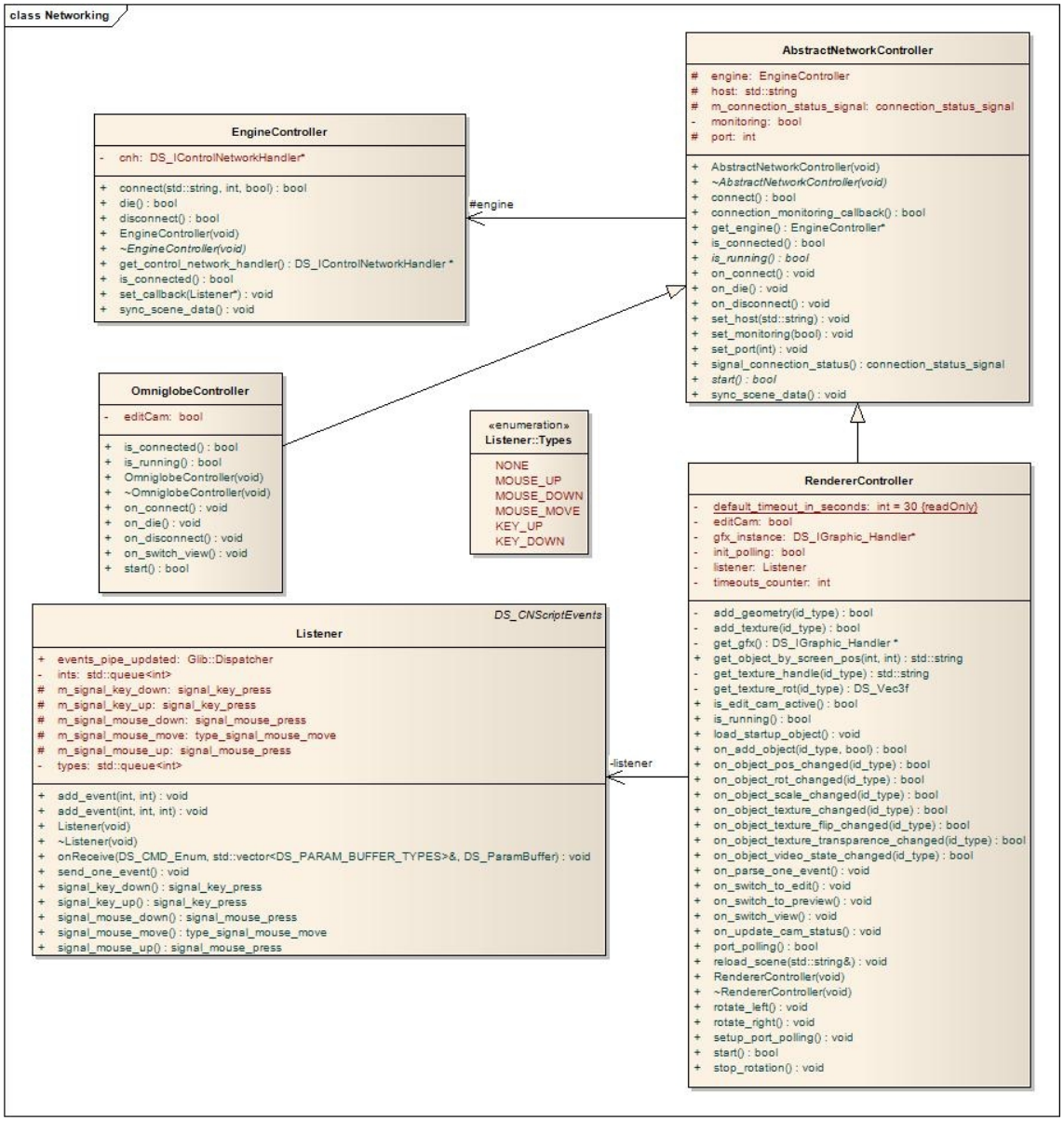

Abbildung 5.6: Klassendiagramm der Netzwerkschicht

- Erstellung und Überwachung der Verbindung mit der Omniglobe/Rendering-Engine,
- Abstrahierung des Zugangs zur Omniglobe/Rendering-Engine,
- Transparenz,
- Kapselung von Multithreading.

Folgende Klassen wurden für die Implementierung dieser Aufgaben erstellt:

AbstractNetworkController Dieser implementiert die Herstellung der Netzwerkverbindung zur 3D-Engine. Auf diesem Niveau wird kein Unterschied zwischen dem Omniglobe und einer Rendering-Engine gemacht. Diese Klasse besitzt die Port- und Hostname-Einstellungen sowie die Instanz der EngineController-Klasse. Die Klasse implementiert die Slots für das Herstellen und den Abbruch der Verbindung. Bei Änderungen des Zustandes der Verbindung wird ebenfalls ein Signal emittiert.

EngineController Diese Klasse hält die DS_IControlNetworkHandler-Instanz, die beim Laden der DLL-Datei erstellt wird. Auf diesem Niveau werden die Low-Level-Einzelheiten der Netzwerkverbindung bearbeitet.

RendererController Dieser implementiert die High-Level-Slots und Signale für die Steuerung der Rendering-Engine. Diese Klasse setzt alle Aufrufe der CSzene-Klasse um, wie z. B. Addition eines Modells, Änderung der Textur usw. Wegen des oben beschriebenen Problems der Netzwerk-Bibliothek wurde diese Klasse auch für den Einsatz mit dem Omniglobe verwendet.

Listener Die Listener-Klasse erledigt die Low-Level-Bearbeitung der Nachrichten, die von der Rendering-Engine ankommen. Diese Klasse implementiert gleichermaßen die Sendung von Signalen zwischen Netzwerk-Thread und GUI-Thread. Dies wird an späterer Stelle noch detaillierter erörtert.

Listing 5.2 präsentiert ein Beispiel der Kommunikation mit der Rendering-Engine. In diesem Beispiel wird die Position einer geometrischen Teilkomponente geändert. Zu diesem Zweck wird erst der Handler das grafische Untersystem erhalten; durch den RPC-Aufruf zur *setPosition*-Funktion wird die Position geändert.

Die ControlNetworkHandler-Bibliothek ist auf Basis der NetTrans-Bibliothek aufgebaut. Für jede Verbindung startet diese Bibliothek einen geeigneten Thread. Aus diesem Grund wurde auch eine Behebung des gleichzeitigen Zugriffs auf Ressourcen realisiert. Die RPC-Aufrufe, wie z. B. im Listing 5.2 gezeigt, werden im GUI-Thread bearbeitet. Die Nachrichten von der Renderer-Engine-Seite werden durch eine Rückruffunktion umgesetzt. Diese

Listing 5.2: Beispiel der Netzwerkkommunikation

```
DS_IGraphic_Handler* RendererController :: get_gfx ()
{
        if ( gfx_instance == NULL) {
            gfx_instance = engine.get_control_network_handler ()->getGraphicHandler();
        }

        return gfx_instance ;
}


bool RendererController :: on_object_pos_changed(id_type id)
{
    EZLOGGERSTREAM << "transmitting new position" << std::endl;

    Plane *obj = (Plane *) Scene.get(id);
    std :: string handle = obj->get_handle();
    if (handle == "") {
        EZLOGGERSTREAM << "Error. Empty handle. Ignoring this operation!" <<std :: endl;
        return false ;
    }

    // get the grafix handler
    DS_IGraphic_Handler *gfx = get_gfx () ;
    if (gfx == NULL) {
        EZLOGGERSTREAM << "Failed to get the GFX-handler." << std::endl;
        return false ;
    }

    if (gfx-> setPosition (handle, obj->get_pos()) == false ) {
        EZLOGGERSTREAM << "Failed to set the position." << std :: endl;
        return false ;
    }
    return true;
}
```

Funktion wird indes im Thread der Netzwerkbibliothek aufgerufen.

Für die Bearbeitung der Rückrufe wurde die Listener-Klasse entwickelt. Im Wesentlichen existiert die Klasse aus folgenden Teilen:

- on_receive Callback Funktion,
- Warteschlange für angekommene Nachrichten,
- Dispatcher,
- durch wechselseitigen Ausschluss (Mutex) geschützte Funktionen für einen Zugriff auf die Warteschlange,
- Signale für Nachrichtentypen:
  - Mouse_Up <int N> : Die N-te Mouse-Taste wurde gedrückt.
  - Mouse_Down <int N> : Die N-te Mouse-Taste wurde freigelassen.
  - Mouse_Move <int X, int Y> : Die Mouse wurde bewegt, neue Koordinaten: <X,Y>.
  - Key_Up <int N> : Die Taste mit dem Scancode N wird gehalten.
  - Key_Down <int N> : Die Taste mit dem Scancode N wurde freigelassen.

Der Dispatcher ist ein spezielles Signal, der aus dem Netzwerk-Thread zum Haupt- oder GUI-Thread geschickt wird. Auf diese Weise wurde der Einsatz von Mutex möglichst stark minimiert.

Die entsprechende Bearbeitung der oben beschriebenen Signale ist damit von der Hardware nahen Ebene effizient getrennt. Die weitere Bearbeitung der Mouse-Ereignissen wird von der MouseTracker-Klasse übernommen, die sich dann lediglich auf die interne Logik des Systems konzentriert.

5.3 Benutzeroberflächenschicht

Die Benutzeroberfläche des Autorentools wurde mithilfe des GTKmm-Frameworks entwickelt. Bei der Entwicklung der Benutzeroberflächen spielt die Anwendung des Signal-Slot-Mechanismus eine wichtige Rolle. Jedes Objekt alleine verkörpert lediglich einen Teil des gesamten Systems, das durch seine Signale und Slots mit anderen Teilen verbunden

wird. Um die Logik der Anwendung zu begreifen, muss man nicht nur das Klassendiagram verstehen, sondern auch die Verbindungen von Klassen durch Signale und Slots, die nur in der Laufzeit der Anwendung entstehen. Abbildung 5.8 illustriert die Anbindung der Slots und Signale in der Anwendung.

Ein Ausschnitt aus der gesamten Systemarchitektur (vgl. Abbildungen 5.9 und 5.10), veranschaulicht die Bestandteile der Benutzeroberflächenschicht.

Gemäß dem Pflichtenheft wurde die Benutzeroberfläche in folgende wesentliche Bestandteile eingeteilt:

- das Hauptfenster - die MainWindow-Klasse,
- das Menü - die MenuHandler-Klasse,
- die Baumansicht der Teilkomponenten - die ItemsTree-Klasse,
- die Werkzeugleiste - die Toolbox-Klasse,
- die Parameter-Frame-Klassen - die ParameterFrame-Klasse und der Baum der Frame-Klassen.

Die MainWindow-Klasse initiiert und bindet alle aggregierten Klassen.

Für die Benutzeroberfläche spielt die gerade ausgewählte Teilkomponente eine bedeutende Rolle. Die Teilkomponente kann in der Baumansicht der Teilkomponenten oder durch Anklicken im Rendering-Engine-Fenster ausgewählt werden. Die Auswahl einer Teilkomponente steuert das Parameter Frame und die Werkzeugleiste. Letztere enthält die Knöpfe für die Addition und das Löschen der Teilkomponenten, die Verbindung mit dem Omniglobe sowie das Erstellen der Szene für den Omniglobe. Die Knöpfe für die Addition und das Löschen der Teilkomponente sind von der gerade ausgewählten Teilkomponente abhängig.

Die Erstellung der Benutzeroberfläche erfolgte gemäß dem Pflichtenheft ohne besondere Schwierigkeiten. Als wichtigster und interessantester Teil der Benutzeroberflächenschicht im Rahmen dieser Arbeit ist der Baum der Parameter Frames zu nennen. Die Einzelheiten der Implementierung und des Designs der anderen Komponenten sind dem beigefügten Quellcode und der Dokumentation zu entnehmen.

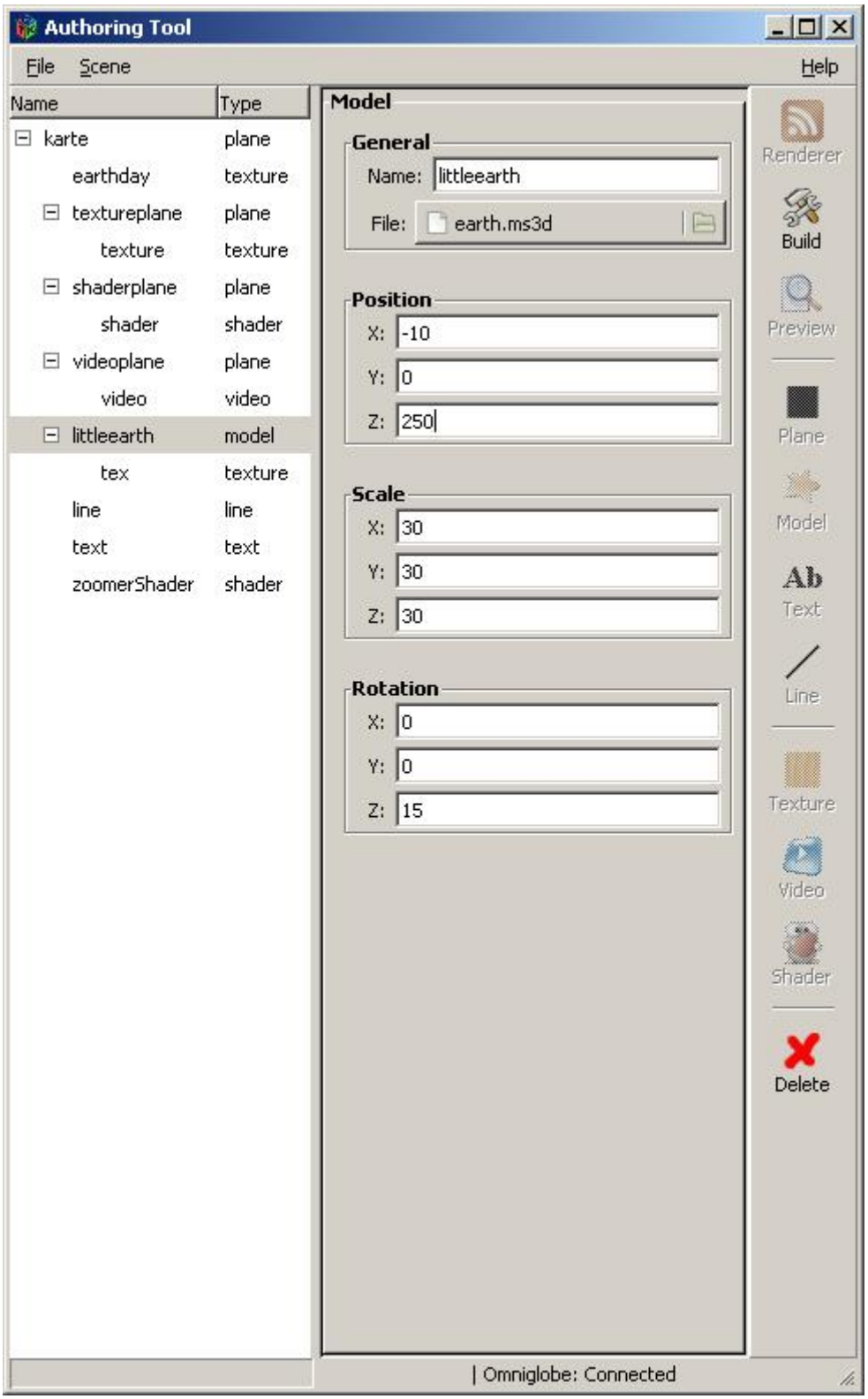

Abbildung 5.7: Benutzeroberfläche des Autorentools

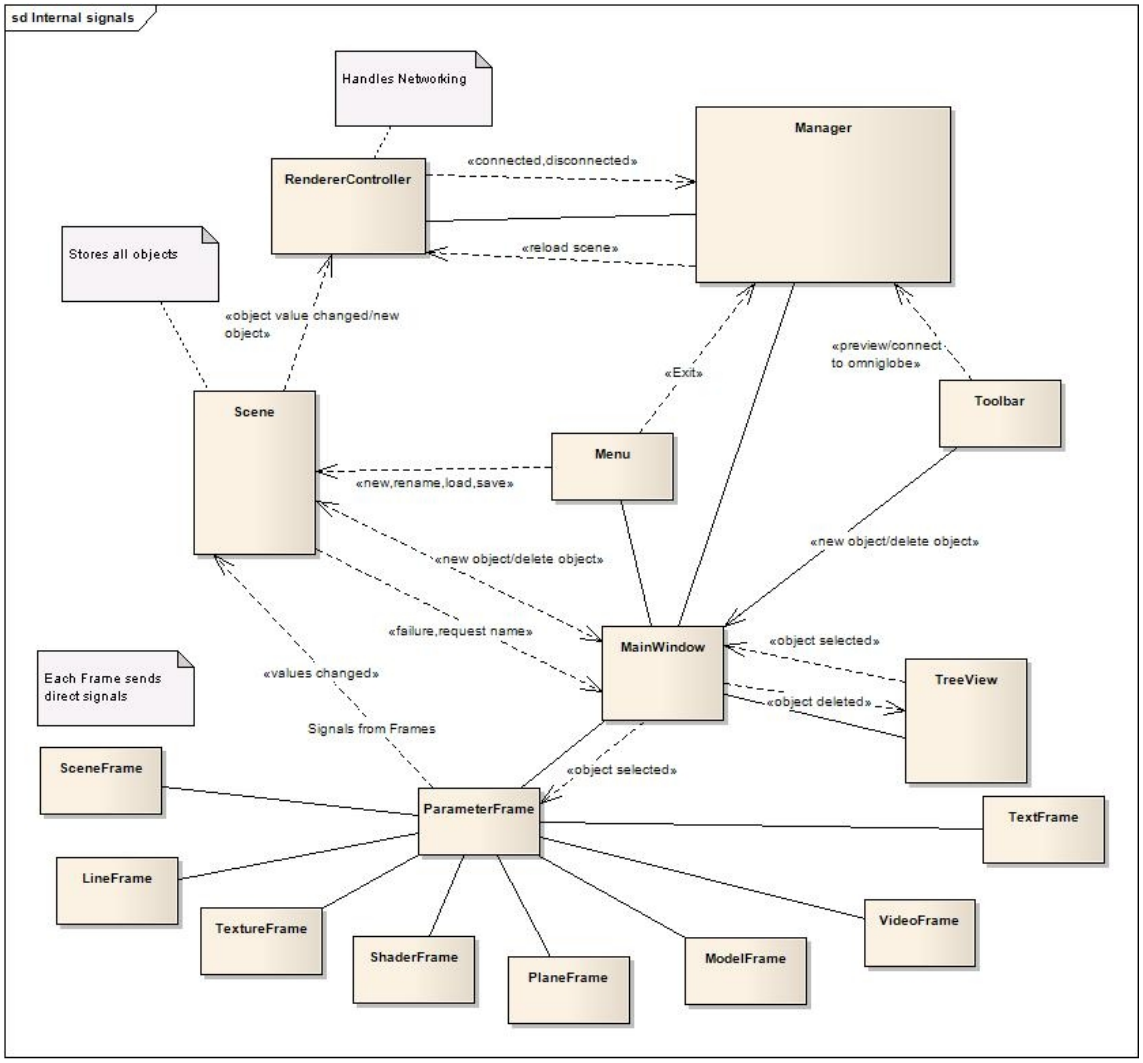

Abbildung 5.8: Verlauf der Signale

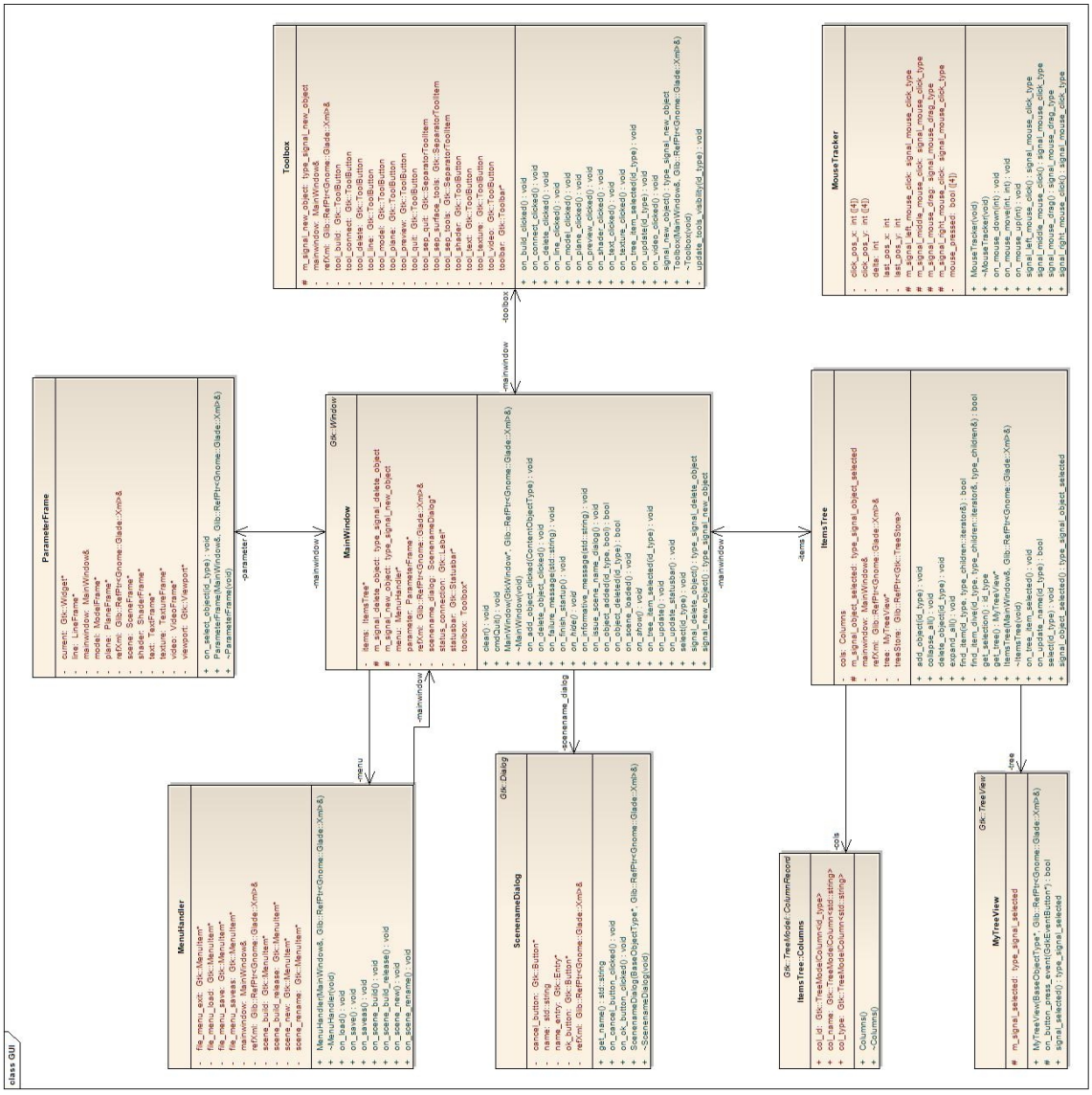

Abbildung 5.9: Klassendiagramm der Benutzeroberflächenschicht

5.3.1 Parameter Frame

Parameter Frames wurden anhand von Glade entwickelt. In Kapitel 3 wurde ein Teil der ParametFrame-Klasse als Beispiel der Anwendung der Libglade-Bibliothek präsentiert. Die Teilkomponenten der Szene besitzen sehr viele ähnliche Parameter, wie z. B. Name, Position, Farbe, Datei, Skalierung und Rotation. Um die Bearbeitung derselben Eigenschaften nicht doppelt zu implementieren, wurde der Baum der Frame-Klassen entwickelt. Im Vergleich zum ContentObject's Baum ist der Baum der Frames komplizierter, da er mehrere Wurzeln aufweist: BaseFrame, FileImporterFrame und ColourFrame. Ferner ist noch die Kompatibilität in den Glade-Dateien zu gewährleisten.

Die BaseFrame-Klasse beispielsweise definiert die Eigenschaft "Name" und die entsprechende Gtk::Entry "name_entry". Diese Gtk::Entry wird in der Glade-Datei gesucht, die dem Konstruktor der BaseFrame-Klasse übergeben wird. Die BaseFrame-Klasse implementiert die Slots für die Bearbeitung der Änderungen, die mit der Gtk::Entry passieren, und das Signal signal_name_changed setzt die Kommunikation mit der CScene-Klasse um. Die Klasse PositionObjectFrame erbt von der BaseFrame-Klasse und definiert dazu entsprechend drei Eigenschaften: Gtk::Entry's, Bearbeitungsfunktionen und ein Signal signal_pos_changed. Keine dieser Klassen besitzt eine Glade-Datei und wird auch nicht als Frame-Klasse auf dem Bildschirm des Benutzers dargestellt. Die reelle Klasse ist in diesem Fall z. B. die TextFrame-Klasse, die von der PositionObjectFrame-Klasse erbt. Die TextFrame-Klasse definiert dazu noch die Eigenschaft "Text" sowie ein entsprechendes Signal und die Gtk::Entry. Dessen ungeachtet erbt die TextFrame-Klasse von einer Wurzel der Frame-Klassen: ColourFrame. Summiert man das Ganze, besitzt die TextFrame-Klasse folgende Eigenschaften und Signale:

- Eigenschaften:
  - ◊ Gtk::Entry* name
  - ◊ Gtk::Entry* text_entry
  - ◊ Gtk::Entry* pos_entry_x, pos_entry_y, pos_entry_z
  - ◊ Gtk::Entry* col_entry_r, col_entry_g, col_entry_b, col_entry_a
- Vorgenannte erwarten folgende Namen in der übergebenen Glade-Datei:
  - ◊ name_entry
  - ◊ text_entry
  - ◊ pos_entry_x, pos_entry_y, pos_entry_z

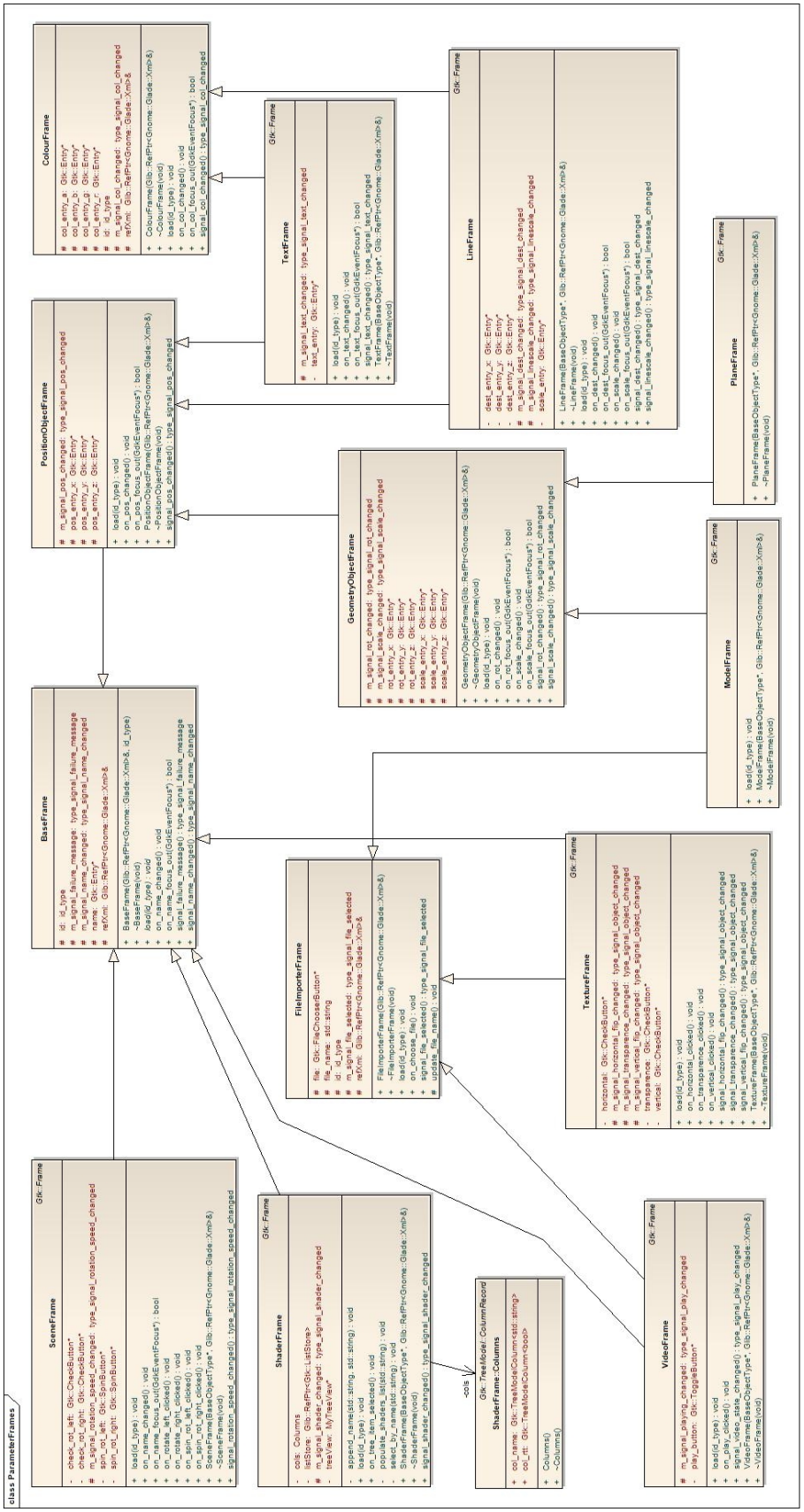

Abbildung 5.10: Klassendiagramm der Parameter Frames

    - ◊ col_entry_r, col_entry_g, col_entry_b, col_entry_a
- Diese wiederum emittieren folgende Signale:
    - ◊ signal_name_changed
    - ◊ signal_text_changed
    - ◊ signal_pos_changed
    - ◊ signal_col_changed

Die TextFrame-Klasse wird von der ParameterFrame-Klasse erzeugt und vorkonfiguriert. Die entsprechende *text.glade*-Glade-Datei, die das Layout der Objekte definiert, wird im Konstruktor der Parameter-Frame-Klasse geladen und als Referenz an den Konstruktor der TextFrame-Klasse übergeben. Der Konstruktor der TextFrame-Klasse gibt die Referenz an die Konstruktoren der oberen Klassen weiter. Wählt der Benutzer eine Text-Teilkomponente in der Baumansicht aus, was beispielweise automatisch beim Erzeugen einer neuen Teilkomponente passiert, wird die Parameter-Frame-Klasse mithilfe eines Signals benachrichtigt. Die entsprechende Funktion lädt die Daten der ausgewählten Teilkomponente in die Instanz der TextFrame-Klasse und präsentiert sie auf dem Parameter Frame im Hauptfenster.

5.4 Dataschicht

Ein Ausschnitt aus der gesamten Systemarchitektur (vgl. Abbildung 5.11) illustriert die Bestandteile der Dataschicht. Die Dataschicht besteht aus einer Sammlung von Klassen, die hauptsächlich für die interne Darstellung der Szene entwickelt wurden. Darunter fallen der Baum der ContentObject-Klassen, welche die Teilkomponenten der Szene implementieren, und die CScene-Klasse. Da die CSzene-Klasse ein Storage und eine Fabrikklasse für den Baum der Teilkomponenten bildet, wurde die Klasse in der Form einer Singleton-Klasse implementiert. Die mögliche Unterteilung der CScene-Klasse in eine Fabrik- und Storage-Klasse sowie eine Scene-Klasse wurde im Rahmen dieser Arbeit als unnötig und nicht produktiv gesehen. Die CScene-Klasse erfüllt folgende Aufgaben:

- Implementierung der Abbildung der Omniglobe-Szene für das Autorentool,
- Speicherung, Verwaltung und Erzeugung der Instanzen der Teilkomponenten (ContentObjects),

- Beinhaltung der Slots für alle Änderungen der Teilkomponenten und Eigenschaften der Szene sowie:
  - on_new_object
  - on_object_..._changed
  - on_build_scene
  - ...
- Implementierung der Methoden, welche die Szene für den Omniglobe erzeugen,
- Methoden für den Export/Import der Projektdateien.

Interne Speicherung und Verwaltung der Daten

Die Teilkomponenten werden intern möglichst abstrakt repräsentiert. Jede Teilkomponente besitzt eine Klasse, die von der ContentObject-Klasse abgeleitet wird. Der dabei entstandene Baum ist Abbildung 5.11 zu entnehmen.

Für die interne Verwaltung der Teilkomponenten und einen schnelleren Zugriff werden gleichzeitig ein assoziatives Array und ein Baum der Referenzen auf die Elemente des assoziativen Arrays eingesetzt. Das assoziatives Array mappt die Identifikatoren der Teilkomponenten zu Teilkomponenten und dient einem schnelleren direkten Zugriff. Dieses Array wird von der CScene-Klasse verwaltet. Jede Teilkomponente enthält ein Array der Kind-Teilkomponenten sowie einen Zeiger auf den Vater. Die Teilkomponente "karte" ist die Wurzel des Baumes und besitzt daher keinen Vater.

5.4.1 Erstellen einer Szene

Der Lebenszyklus des Autorentools ist in Abbildung 5.12 illustriert. Der auf der Abbildung gezeigte "build-reload"-Zyklus bildet den Kern der Anwendung und wird sehr häufig verwendet. In vielen Fallen, so wie z. B. bei der Addition einer Teilkomponenten oder Änderung der Parameter, ist die gewünschte Änderung nicht ohne Erstellung der Szene und das Laden der neuen Szene nicht möglich. Der "build-reload"-Zyklus repräsentiert ebenfalls den Kern hinsichtlich der Erstellung der Szene für den Omniglobe. Es existieren zwei unterschiedliche Varianten der Szene: die Szene für die weitere Bearbeitung und die Szene für die Darstellung auf dem Omniglobe. Die zweite Variante wird in der Dokumentation

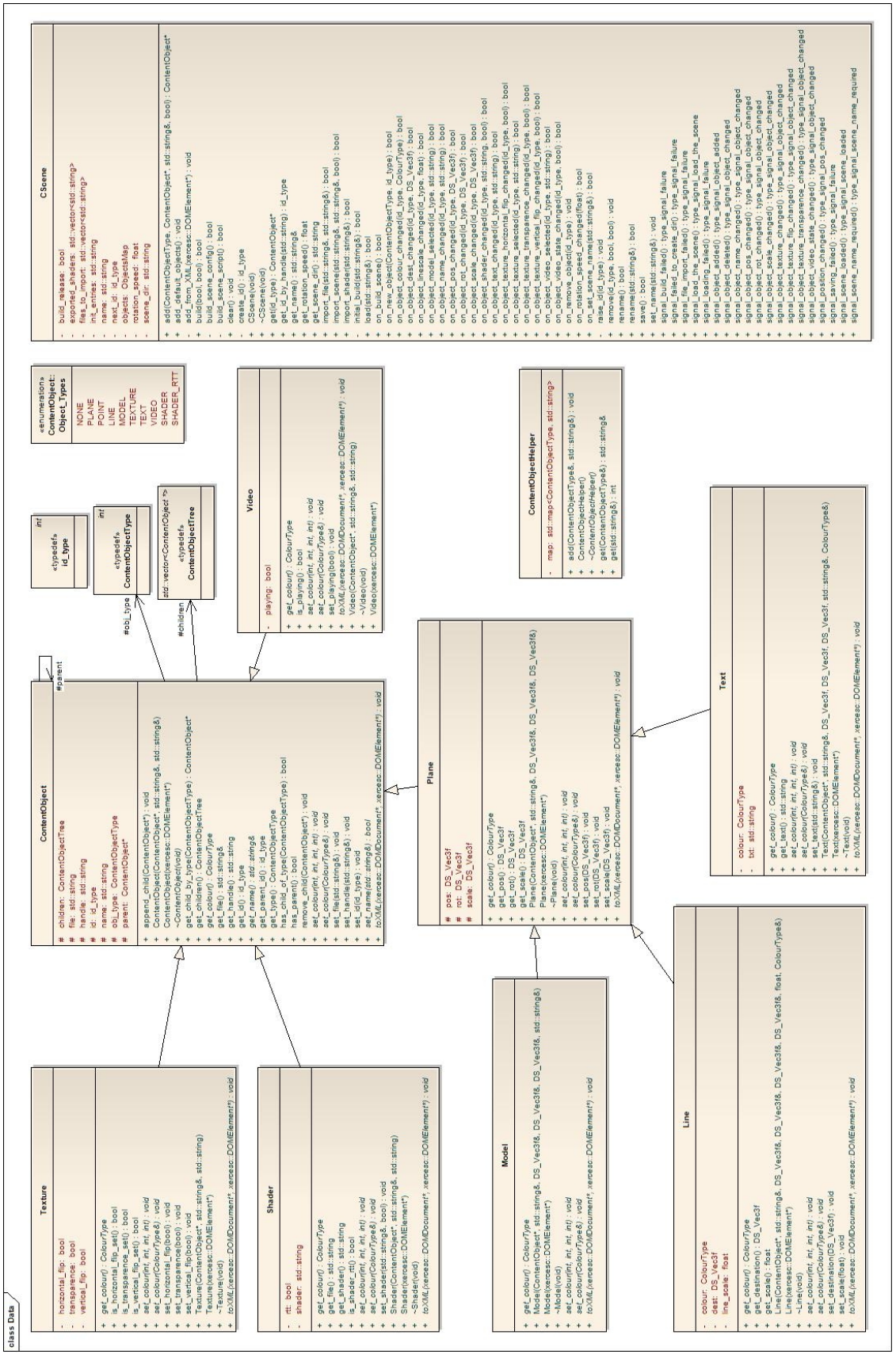

Abbildung 5.11: Klassendiagramm der Dataschicht

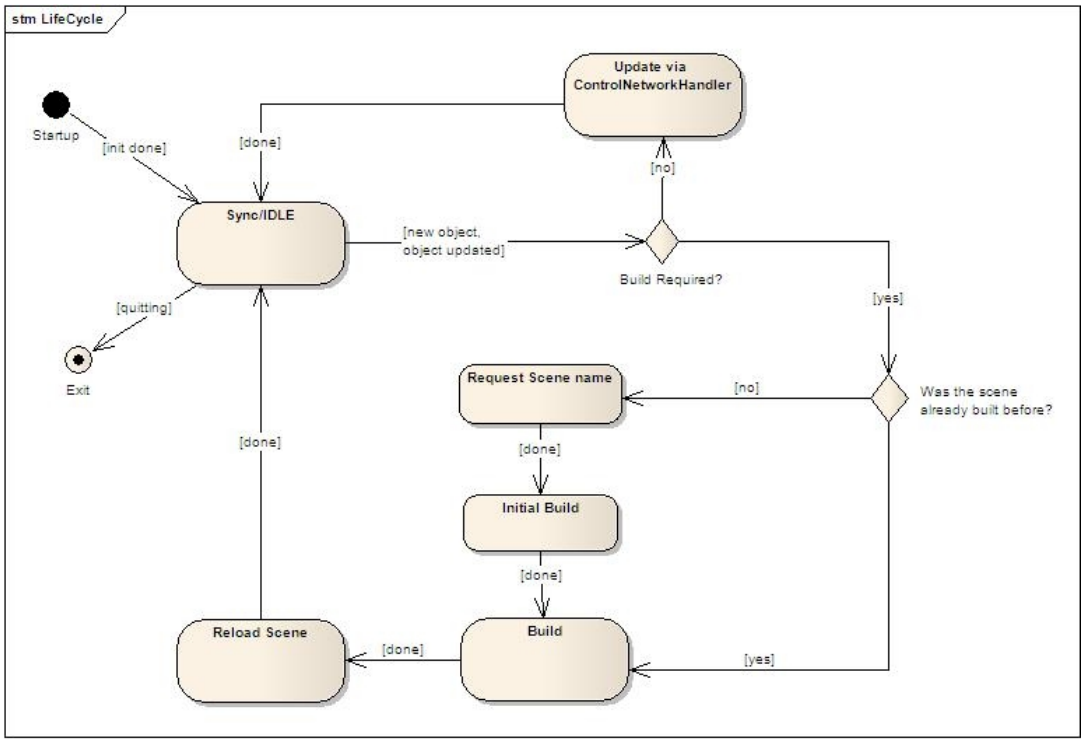

Abbildung 5.12: Lebenszyklus des Autorentools

als „Release“ bezeichnet. In Abschnitt 5.1.2 wurden einige Unterschiede zwischen beiden Varianten erklärt.

Die Erstellung der Szene besteht aus folgenden Grundoperationen:

1. Erstellung der Grundszene, falls noch nicht vorhanden:
   - Name der Szene nachfragen.
   - Szeneordner erstellen.
   - Grunddateien aus dem Szeneordner der “Authoring“-Szene importieren.
2. Erstellung der Szene-XML-Konfigurationsdatei:
   a) Die Parameter der Szene einfügen.
   b) Die Komponenten der Szene und Skripten durch <Include>- und <ScriptInclude>-Befehle einfügen.
   c) Die Callback-Funktionen für die Hauptskriptdatei definieren.
   d) Die Liste der Teilkomponenten umgehen:
      i. 3D-Modelle und Ebene als <GeometricElement> einfügen und Kinder davon prüfen:

A. Falls eine Textur ”Kind“ existiert:
   - Die Textur zur geometrischen Teilkomponente addieren.
   - Falls die Textur extra Parameter, wie etwa Spiegelung, besitzt, merken (*).

B. Falls ein Shader-Kind vorliegt:
   - Die Shader-XML-Konfigurationsdatei einfügen.
   - Name des Shaders zur geometrischen Teilkomponente addieren.

C. Falls ein Video-Kind existiert:
   - VideoCapture-Abschnitt einfügen.
   - Das Video merken (*).

ii. Text wird als eine unsichtbare, beschriftete geometrische Teilkomponente addiert.

iii. Linie merken (*).

3. Erstellung der Skriptdatei:

   a) Die globalen Variablen definieren.

   b) Die *init*-Funktion definieren.

   c) Die Bearbeitung der mit (*) gezeichneten gemerkten Teilkomponenten einfügen.

   d) Die *main*-Funktion definieren.

In Abbildung 5.13 ist der Callgraph der ”build“-Methode der CSzene-Klasse veranschaulicht. Die Struktur der Szene-XML-Konfigurationsdatei wurde n Kapitel 3 erläutert.

5.4.2 Export und Import der Projektdatei

Die Projektdatei beschreibt möglichst minimalistisch den inneren Zustand der Autorentools zum Zeitpunkt der Abspeicherung. Die Projektdatei enthält lediglich den Namen der Szene und den Baum der Teilkomponenten, aus denen die Szene besteht. Um die Konsistenz mit der entwickelten Szene zu gewährleisten, wird nach dem Speichern der Projektdatei stets der ”build-reload“-Zyklus ausgeführt. Für die Abspeicherung wird der Baum der Teilkomponenten benutzt. Die einzelnen toXML- und fromXML-Funktionen sind in der ContentObject-Klasse implementiert und werden in einer Kette aufgerufen. Aus dem Aufruf Linie.toXML beispielsweise resultiert ein Aufrufen zu Plane.toXML und ContentObject.toXML. Jede Klasse exportiert damit nur die Daten, welche die Klasse aufbewahrt. Ein Beispiel der Projektdatei ist aus Abbildung 5.3 ersichtlich.

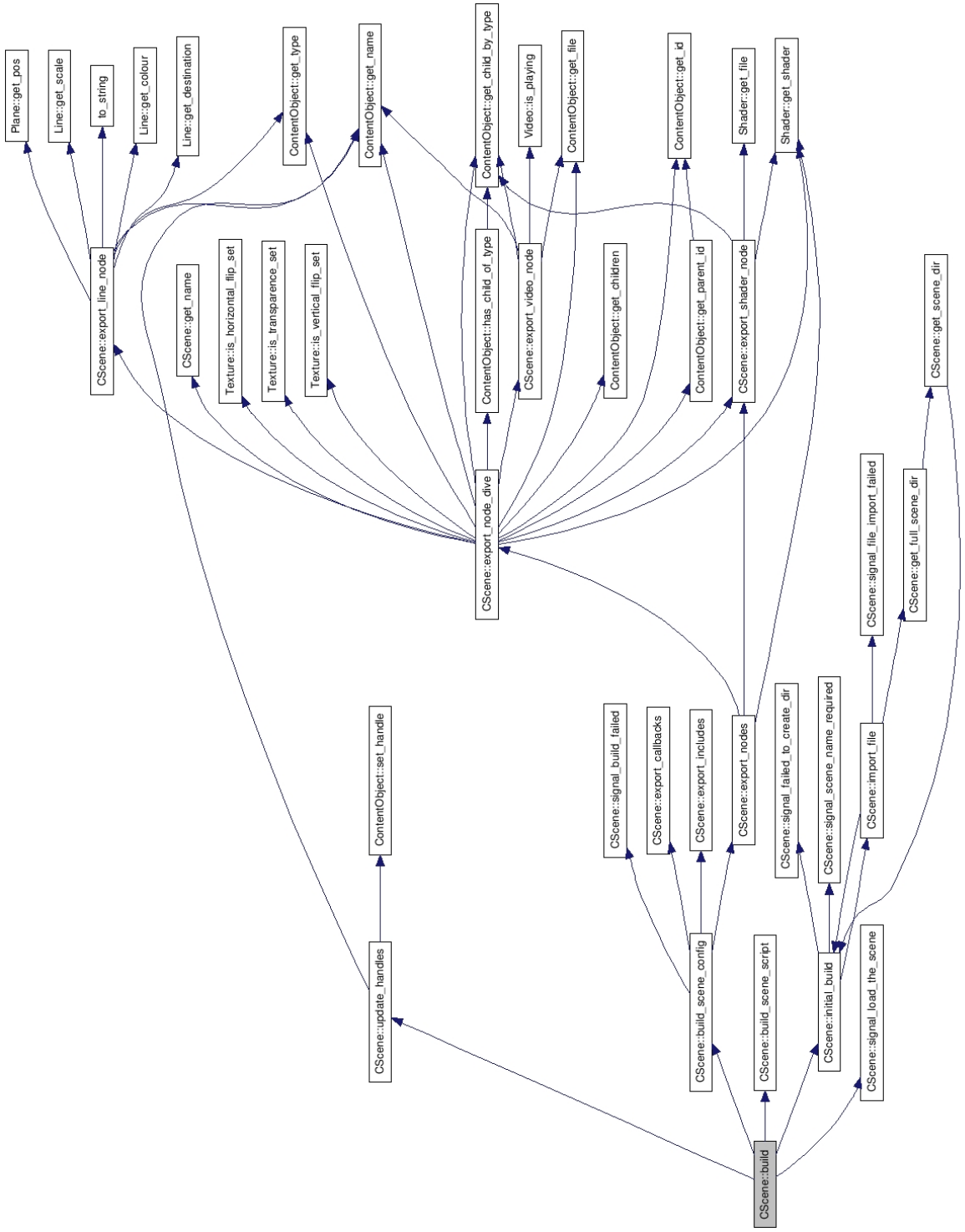

Abbildung 5.13: Callgraph der build Methode

Listing 5.3: Beispiel der Projektdatei

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<AToolProject name="presentation" rotation_speed="-0.2">

  <Object file="" id="1" name="karte" parent_id="0" pos="0 -70.3125
-21.3438" rot="0 0 0" scale="1 1 1" type="plane">
    <Object file="EarthDay_f_bright.jpg" flipH="true" flipV="false"
id="2" name="earthday" parent_id="1" type="texture"/>
    <Object file="" id="3" name="textureplane" parent_id="1" pos="-5 0
-600" rot="0 0 0" scale="1 1 1" type="plane">
      <Object file="default.png" flipH="true" flipV="false" id="9"
name="texture" parent_id="3" type="texture"/>
    </Object>
    <Object file="" id="6" name="shaderplane" parent_id="1" pos="-5 0
600" rot="0 0 0" scale="1 1 1" type="plane">
      <Object RTT="false" Shader="CityNoise" file="default.png" id="10"
name="shader" parent_id="6" type="shader"/>
    </Object>
    <Object file="" id="8" name="videoplane" parent_id="1" pos="-5 0 0"
rot="0 0 0" scale="1 1 1" type="plane">
      <Object file="Meeting.wmv" id="11" name="video" parent_id="8"
playing="true" type="video"/>
    </Object>
    <Object file="earth.ms3d" id="12" name="littleearth" parent_id="1"
pos="-10 500 0" rot="0 0 15" scale="30 30 30" type="model">
      <Object file="huge.jpg" flipH="true" flipV="false" id="13"
name="tex" parent_id="12" type="texture"/>
    </Object>
    <Object colour="255 100 255 255" dest="-10 -400 800" file="" id="14"
linescale="4" name="line" parent_id="1" pos="-10 -400 -800" rot="0 0 0"
scale="0.01 0.01 0.01" type="line"/>
    <Object colour="150 255 100 100" file="" id="15" name="text"
parent_id="1" pos="-10 -350 0" rot="0 0 0" scale="0.01 0.01 0.01"
text="Presentation Scene" type="text"/>
    <Object RTT="true" Shader="Zoomer" file="default.png" id="16"
name="zoomerShader" parent_id="1" type="shader"/>
  </Object>

</AToolProject>
```

Kapitel 6

Bewertung und Ausblick

In diesem Kapitel wird das entwickelte System evaluiert und bewertet. Abschnitt 6.1 besteht aus der Durchführung der Tests aus dem Pflichtenheft. In den nachfolgenden Abschnitten 6.2 und 6.3 werden die Ergebnisse der Tests besprochen, bevor ein Vergleich mit der Lage vor dem Einsatz des Autorentools stattfindet. Auf der Grundlage dieser Abschnitten erfolgt in Abschnitt 6.4 eine resultierende Gesamtbewertung des Systems. Abschließend werden in Abschnitt 6.5 die Verbesserungsvorschläge und Ideen für die Weiterentwicklung des Autorentools vorgestellt.

6.1 Tests

In Addition zu den Unit-Test der einzelnen Klassen, wurden alle Produktfunktionen aus dem Pflichtenheft einzeln getestet. Zu diesem Zweck wurden folgende Schritte unternommen:

/T0001/ Starten der Anwendung,

/T0002/ Erzeugung einer neuen Szene,

/T0003/ Umbenennung der Szene,

/T0004/ Addition von drei Ebenen,

/T0005/ Umbenennung der Ebenen zu: “textureplane”, “videoplane”, “shaderplane”,

/T0006/ Skalierung, Positionierung, Rotierung und Umbenennung der drei Ebenen,

/T0007/ Addition eines Modells,

/T0008/ Skalierung, Positionierung, Rotierung und Umbenennung des Models,

/T0009/ Umbenennung des Modells zu: “littleearth”,

/T0010/ Addition eines Videos zu der “videoplane”-Ebene,

/T0011/ Addition einer Textur zu der “texturplane” Ebene,

/T0012/ Addition eines “CityNoise”-Shader zur “shaderplane”-Ebene,

/T0013/ Addition einer Linie,

/T0014/ Änderung von Position, Farbe, Name, Dicke,

/T0016/ Addition einer Text-Teilkomponente,

/T0017/ Positionierung, Änderung der Farbe, Umbenennung, Änderung des Textes,

/T0018/ Addition eines “Zoomer”-RTT-Shader.

Jeder Test enthielt ferner folgende „Untertests“:

/T0030/ Realvorschau,

/T0031/ Erstellung der Release-Version der Szene,

/T0032/ Export der Szene zum Omniglobe,

/T0033/ Abspeicherung in einer Projektdatei,

/T0034/ Laden aus der Projektdatei,

/T0034/ Löschen der Teilkomponenten nach jeder Änderung.

Darüber hinaus wurde auch die Erweiterbarkeit des Systems durch Einführung der Animatoren geprüft. Alle Tests konnten erfolgreich und den Erwartungen entsprechend abgeschlossen werden. Als Ergebnis der Tests ist eine “Presentation”-Szene entstanden, die auf den Bildern 6.1 und 6.2 zu sehen ist.

Abbildung 6.1: “Presentation”-Szene im Bearbeitungsmodus

Abbildung 6.2: “Presentation”-Szene auf dem Omniglobe

6.2 Ergebnisse

Alle Musskriterien sind gemäß den aktuellen Anforderungen vollständig realisiert. Dessen ungeachtet wurden vier Wunschkriterien erfüllt: Abspeicherung und Laden der Projektdateien, erweiterte Unterstützung der HLSL-Effekte, die Möglichkeit, die Szene zu erweitern und die Möglichkeit die Szene in Echtzeit auf dem Omniglobe darstellen und bearbeiten. Das letzte Wunschkriterium wurde aus technischen Gründen nicht implementiert.

Aus technischen Gründen, die in Kapitel 5 bereits zum Teil Erwähnung fanden, wurden folgende Abweichungen zu den Anforderungen des Pflichtenheftes bei der Entwicklung des Systems festgestellt:

- Das DevDelight-Framework ist eine für Echtzeit-Rendering ausgestattete Engine, die für die Performance abgestimmt ist. Aus diesem Grund wird beim zweiten Laden einer Textur immer die Kopie benutzt, die bereits geladen ist. Daraus resultiert ein gemeinsames Übertragen der Textureinstellungen, wie z. B. Spiegelung um die Achse. Wird eine Textur zweimal an geometrischen Teilkomponenten angewandt und bei “einer” davon die Spiegelung gesetzt, wird die Spiegelungseinstellung auch an die “zweite” Textur übergeben.
- Die Linien-Teilkomponenten besitzen keine Parent-Teilkomponenten. Bei der Im-

plementierung der Linie-Teilkomponenten wurde festgestellt, dass die Linie eine vollständig geskriptete Teilkomponente bildet und somit nicht zu den Teilkomponenten, die auf dem <GeometricElement>-Tag der Szene-XML-Konfigurationsdatei basieren, gehört. Damit besitzt diese Teilkomponente keine Parents und Positionsparameter, die von der Rendering-Engine automatisch nachgerechnet werden. Durch eine besondere Änderung der Skripten wird die Position jeder Linie relativ zur "karte"-Teilkomponente berechnet.

Mit diesen zwei Ausnahmen wurden die Anforderungen des Pflichtenheftes erfolgreich realisiert. Generell wurden im entwickelten System folgende Nachteile entdeckt:

- Eine CPU des Systems wird vom Rendering-Engine-Prozess immer zu 100% ausgelastet. Aus diesem Grund ist die Benutzung des Autorentools auf einem System ohne mehreren CPU-Kernen oder zumindest HyperThreading nicht fließend genug.
- Der "build-reload"-Zyklus wird an sehr vielen Fallen ausgeführt. Beispielweise sind lediglich circa 10% der Änderungen der Parameter der Teilkomponenten durch den direkten Einsatz der ControlNetworkHandler möglich. Die restlichen Änderungen werden immer durch den "build-reload"-Zyklus ausgeführt.
- Die Parameter Frames sind hinsichtlich des frequenten "build-reload"-Zyklus unpassend projektiert.

6.3 Vergleich vorher/nachher

Vorher repräsentierte die Entwicklung der Szenen für den Omniglobe einen nicht trivialen Prozess. Der Entwickler sollte Kenntnisse in den Bereichen der 3D-Modelling und Rendering-Software besitzen sowie die internen Abläufe des DevDelight-Frameworks und die Struktur der Szene kennen. Die Szenen wurden rein durch die "trial and error"-Methode ohne Vorschau erstellt. Um die entstandene Szene zu testen, sollte man am besten den Omniglobe besitzen, um die Szene auf diesem betrachten zu können. Die Entwicklung bestand aus mehreren kleinen Operationen und sich wiederholenden Testläufen.

Außerdem forderte die Struktur der Szene eine lange Einarbeitungsphase. Verschiedene Komponenten des Systems waren in einer Skriptdatei verpackt. Im Rahmen der Umstrukturierung der Szene, die im Rahmen dieser Arbeit durchgeführt wurde, und der Einführung von Multiple-Callback-Funktionen wurden die Komponenten der Szene auseinan-

dergenommen und in Module gekapselt. Dies hat allein die Verständlichkeit der Szenen und der wiederverwendbaren Komponenten deutlich verbessert.

Durch die Anwendung des Autorentools wird die Entwicklung von Szenen deutlich erleichtert. Eine Zusammenfassung ist in Tabelle 6.3 gegeben.

Aufgabe	**vorher**	**nachher**
Erstellung einer neuen Szene	10-20 Minuten	1 Minute
Hinzufügen der Elemente	bis 1 Stunde	1 Minute
Transport zum Omniglobe	5-10 Minuten	1 Minute
Vorschau	nicht vorhanden	1 Knopfdruck
Anwendung der vorbereiteten Shader	10-20 Minuten	1 Minute
Erstellung neuer Shader	nicht verändert	nicht verändert
Erstellung komplexer Skripten	nicht verändert	nicht verändert
Animation	nicht verändert	nicht verändert
Einsatz ohne Omniglobe	ungenau, kompliziert	Realvorschau

Tabelle 6.1: Vergleich vorher/nachher

6.4 Bewertung

Das System wurde erfolgreich entwickelt und in Betrieb genommen. Aus der Sicht des Benutzers hat die entwickelte Anwendung die Entwicklung der Szenen für den Omniglobe trotz einiger technischer Nachteile deutlich verbessert und erleichtert. Aus der Zusammenfassung, die in Abschnitt 6.3 gegeben wurde, wird deutlich, dass die entwickelte Software Vorgänge wie Erstellung, Positionierung und Konfigurierung geometrischer Teilkomponenten automatisiert und erleichtert. Die komplexeren Tätigkeiten, wie Programmierung der Shader, Komponenten des Systems, die Animation der existierenden Teilkomponenten, sind durch den Einsatz des Autorentools und neuer Szenenstrukturen gleichsam betroffen. Die Modularisierung der Szene erlaubt eine gezieltere und leichtere Entwicklung der Komponenten sowie eine Inbetriebnahme.

Abgesehen von den komplizierten Fällen bei der Fehlerbehebung im Zusammenhang mit der ControlNetworkhandler-Bibliothek gab es keine wesentlichen Schwierigkeiten bei der Umsetzung des Systems. Der Einsatz einer Netzwerk-gesteuerten Rendering-Engine hat sich gelohnt und als sinnvoll erwiesen.

6.5 Weiterentwicklung des Systems

Das im Zuge dieser Abhandlung entwickelte Softwaresystem stellt eine funktionsfähige Anwendung dar. Der Spruch ”Eine gute Software wird nie fertig“ deutet indes auf jederzeit vorhandene Erweiterungsmöglichkeiten hin. Nach der erfolgreichen Testphase entstanden solche ebenfalls hinsichtlich der Funktionalität des Autorentools.

Benutzerfreundlichkeit

Die Verbesserung der Benutzerfreundlichkeit und die Erweiterung der Funktionalität der ControlNetworkhandler-Bibliothek sind als Erstes zu bewältigen. Der ”build-reload“-Zyklus sollte an möglichst wenigen Stellen ausgeführt werden. Die Anzahl von Aufrufen lässt sich durch die Einführung einiger Funktionen in die ControlNetworkHandler-Bibliothek reduzieren. Alleine die Einführung der setParent-Funktion und die Möglichkeit, die Skripten oder Funktionen in Laufzeit zu erzeugen und auszuführen, hätte etwa drei Vierte der Aufrufe zum ”build-reload“-Zyklus gespart.

Folgende Änderungen könnten die Benutzerfreundlichkeit des Autorentools verbessern:

- Ein spezieller Rendering-Modus, in dem die Rendering-Engine mit einer festen Bildrate (engl.: Frames Per Second) funktioniert. Damit wird die CPU nicht immer zu 100% ausgelastet.
- Bessere Unterstützung für die Mouse im Engine-Renderer-Fenster für die Operationen, wie z. B. Positionierung von Teilkomponenten.
- Die Parameter Frames könnte man mit einem ”Apply“-Knopf erweitern. Die Änderungen werden dann nur durch Knopfdruck an die Rendering-Engine übergeben. Damit wird die Anzahl der ”build-reload“-Zyklen deutlich verkleinert.
- Die Geometrie der Szene ändern, sodass die Karte in der X-Y-Ebene liegt.

Realvorschau

Erstrebenswert ist ferner eine Verbesserung der Realvorschau. Das Bild wird auf den Omniglobe nicht linear gemappt. Das heißt, dass der Realvorschaumodus die Szene etwas an-

ders präsentiert als der Omniglobe. Eine zu 100% genauere Darstellung wäre mithilfe eines Shader, der das Mapping des Omniglobe nachahmt, möglich.

Verbesserung der Bildqualität

Die Halbsphäre entspricht der Transformation des Oberspiegels nicht exakt. Dadurch entstehen z. B. Probleme bei der Kalibrierung für die Ausgabe auf dem Omniglobe und Verzerrungen. Dieses Problem kann man durch einen Austausch der Halbsphäre gegen ein elliptisches Modell lösen.

Eine bessere und flexible Möglichkeit besteht im Einsatz eines SzeneShader. Szene-Shader sind Shader, die das aus dem Rendering resultierende 2D-Bild bearbeiten. Mithilfe dieser Szene-Shader werden die Ausgabebilder der Projektoren zusätzlich verzerrt. Ferner ist in diesem Kontext bemerkenswert, dass die Parameter der Verzerrung in Echtzeit änderbar sind. Auf diese Weise ist die Kalibrierung des Ausgabebildes möglich. Ein Szene-Shader bietet darüber hinaus die Möglichkeit der interaktiven Erstellung der Maske, welche die doppelt belichteten Bereiche des Bildes abschneidet.

Geometrie der Szene

Mit dem Übergang zum Echtzeit-Rendering wurde die Qualität des Bildes verschlechtert. Es existieren bestimmte Bildartefakte, die der geometrischen Struktur der Szene geschuldet sind. Befindet sich ein Objekt beispielweise nicht direkt im Zentrum der Karte, sondern fast an der linken oder rechten Grenze, dann wird auch diese Seite des Objektes angezeigt, da die Kamera etwas von der Seite sieht. In diesem Fall wird das Objekt auch von diesem Blickwinkel aus auf dem Omniglobe dargestellt. Diese Bildartefakte sind lediglich durch Änderungen des grundlegenden geometrischen Aufbaus der Szene möglich. Eine Alternative bestünde darin, anstatt der flachen Karte einen Zylinder zu verwenden. Statt einer Kamera muss man in diesem Fall zwei Kameras mit Gesichtsfeld (engl.: Field of View) von 180° einsetzen. Die beiden Bilder werden auf die Halbsphäre mithilfe eines Shader gemappt.

Vorteile dieser Struktur:

- treues Ausgangsbild,

- keine Probleme der sich überscheidenden Kanten (linke und rechte Kante müssen passen).

Als enormer Nachteil dieser Struktur ist die erheblich steigende Komplexität der Szene zu nennen. Alle flachen Objekte und Teilkomponenten der Szene sind nicht mehr einsetzbar. Beispielsweise müssen die Linien zu Spiralen umgewandelt werden. Die flachen Ebenen müssen gekrümmte Modelle benutzen, die gleichzeitig die Qualität verschlechtern und die Anzahl der Polygonen in der Szene steigern lassen. Sogar die Umrechnung der Geokoordinaten ist in diesem Fall ein dreidimensionales Problem.

Kapitel 7

Abschließende Betrachtung

Abschnitt 7.1 diese Kapitels enthält einen ausführlichen Erfahrungsbericht des Autors dieser Diplomarbeit. Der nachfolgende Abschnitt 7.2 besteht aus einer abschließenden Zusammenfassung der gesamten Arbeit.

7.1 Erfahrungsbericht

Bei der Entwicklung der Anwendung mussten weitere Punkte bearbeitet werden. Beim Einsatz der GTKmm-Open-Source-Bibliothek im Microsoft Windows Betriebssystem kam es zu verschiedenen Kompatibilitätsproblemen der verschiedenen Versionen der Open-Source- Produkte. Die Bibliotheken für Microsoft Windows waren veraltet und nicht vollständig. Diese Schwierigkeiten konnten nur mit dem Release einer neuen Version der GTKmm-Bibliothek gelöst werden. Einige Kompatibilitätsprobleme sind ferner durch Anwendung des Microsoft C++-Kompilers entstanden.

Der Signal-Slot-Mechanismus hat sich als eine gute Methode erwiesen, die aber gleichsam Gefahren birgt. Anhand der Benutzung dieses Mechanismus lassen sich Schnittstellen der Klassen gut definieren. Die Signale und Slots müssen indes verbunden werden, da erst die Verbindungen die Logik der Anwendung aufbauen. Das Problem besteht darin, dass nicht sofort klar ist, was nach dem Emittieren des Signals geschieht. Welche Methode aufgerufen wird, wird erst in der Laufzeit der Anwendung definiert. Der Signal-Slot-Mechanismus ist somit eine sehr mächtige und flexible Programmiertechnik, die aber nur mit gutem Design der Software zu guten Ergebnissen führt.

Der Einsatz der ControlNetworkHandler-Bibliothek ist nicht ohne Komplikationen verlaufen. Die Bibliothek war noch unreif und wurde nicht ausführlich getestet. Die Komplexität des DevDedlight-Frameworks, eine große Anzahl von Interfaces sowie Kompatibilitätsprobleme haben die Fehlersuche sehr problematisch gestaltet. Darüber hinaus war der Entwickler der Bibliothek in den vergangenen drei Monaten sehr beschäftigt, weshalb einige Fehler noch immer nicht korrigiert werden konnten.

7.2 Zusammenfassung

Durch diese Diplomarbeit wurde ein Autorentool für das bereits existierende sphärische Projektionssystem entwickelt, mit dessen Hilfe man das Content für die Darstellung entwickeln kann. Das Projektionssystem setzt eine 3D-Rendering-Engine ein, die in Echtzeit die Ausgabebilder für die Projektoren generiert.

Die Abhandlung umfasst alle Phasen der Softwareentwicklung von der formalen Anforderungsdefinition und Spezifikation über den Entwurf sowie die Implementierung bis zum Test und zur Dokumentation.

Das System ermöglicht auf einfache Art und Weise, die Entwicklung der Szenen für die Präsentation auf dem Projektionssystem zu beschleunigen. Dem Benutzer steht eine Auswahl der vorgefertigten Teilkomponenten zur Verfügung, die den Szenen hinzugefügt werden können. Für die leichtere Bearbeitung wird die Szene als Realvorschau angezeigt und bei Bedarf im Erstellungsprozess auf dem Omniglobe dargestellt.

Das System basiert auf der Rendering-Engine des Projektionssystems, die durch eine Netzwerkanbindung gesteuert wird. Die Addition von Teilkomponenten wird meistens durch Erstellung und Wiederladen der Szene in der Rendering-Engine erledigt. Als Ergebnis der Arbeit erhält man eine vollständige Szene, die man nach dem Wunsch des Benutzers automatisch auf den realen Omniglobe übertragen kann. Der Export einer Projektdatei für die spätere Wiederbearbeitung der Szene ist ebenfalls möglich. Die Automatisierung der Entwicklung bedingt große Einsparungen hinsichtlich der zeitlichen und personellen Ressourcen.

Literaturverzeichnis

Alexandrescu 2001

Alexandrescu, Andrei: *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, 2001 (ISBN 0201704315)

Balzert 1998

Balzert, Helmut: *Lehrbuch der Software-Technik*. Spektrum, 1998 (ISBN 3-8274-0065-1) 9

Breymann 1997

Breymann, Ulrich: *C++ Eine Einführung*. 4. Aufl. Carl Hanser Verlag, 1997

Brooks 1995

Brooks, Frederick P.: *The Mythical Man Month*. 20th Anniversary Edition. Addison-Wesley Longman, 1995 (ISBN 0201835959) 6

Committee 2004

Committee, IEEE Computer Society Professional P.: *Guide to the Software Engineering Body of Knowledge*. Angela Burgess, 2004 (ISBN 0-7695-2330-7)

Freeman und Freeman 2006

Freeman, Eric ; Freeman, Elisabeth: *Entwurfsmuster von Kopf bis Fuß*. O'Reilly Verlag GmbH & Co. KG, 2006 (ISBN-10 3-89721-421-0) 11

Leiterman 2004

Leiterman, James: *Learn Vertex and Pixel Shader Programming with DirectX 9*. Wordware Publishing, Inc., 2004 (ISBN 1556222874)

Lyman und Varian 2003

Lyman, Peter ; Varian, Hal R.: HOW MUCH INFORMATION 2003? (2003). http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/ 1

Putnam und Myers Nov

Putnam, L. ; Myers, W.: How Solved Is the Cost Estimation Problem? In: *IEEE Software* (Nov. 1997), S. pp. 105–107. 10

Roger S. Pressman 2001

Roger S. Pressman, Ph.D. ; Jones, Betsy (Hrsg.): *Software Engineering: A Practitioner's approach*. 5th ed. McGraw-Hill, 2001 (ISBN 0073655783) 8

Shalloway und Trott 2001

Shalloway, Alan ; Trott, James: *Design Patterns Explained: A New Perspective on Object-Oriented Design*. 1st ed. Addison-Wesley Professional, 2001 (ISBN 0201715945)

Anhang A

Anhang

Beispiel der Szene XML-Konfigurationsdatei

Listing A.1: Beispiel der Szene XML-Konfigurationsdatei

```
<?xml version="1.0" encoding="ISO8859-1"?>
<Scene activeCamera="preview" activeFog="" activePlaylist ="" activeSkybox=""
    bgcolor="0 0 0 1" guiAlpha="200" shadowColor="0 0 0 1" soundTrackedNode="camera"
    loop="true" name="authoring" time="10">

    <ScriptInclude section=" first " path="./ script_earth_planes .dscr"/>
    <ScriptInclude section="manager" path="Includes/ Scripts /Manager.dscr"/>
    <ScriptInclude section="keydown" path="Includes/ Scripts /KeyDown.dscr"/>
    <ScriptInclude section="zoomer" path="Includes/ Scripts /ZoomerMod.dscr"/>
    <Include file =" ./ karte_bw.xml"/>
    <Include file =" ./ Includes /Configs/cam.xml"/>

    <ScriptCallback type="kd" function="keyDown"/>
    <ScriptCallback type="ku" function="keyUp"/>
    <ScriptCallback type="playAs" function="playSong"/>
    <ScriptCallback type="stopAs" function="stopSong"/>
    <ScriptCallback type="pauseAs" function="pauseSong"/>
    <ScriptCallback type="mm" function="mouseMove"/>
    <ScriptCallback type="mbd" function="mouseButtonDown"/>
    <ScriptCallback type="mbu" function="mouseButtonUp"/>
    <ScriptCallback type="mw" function="mouseWheel"/>
    <ScriptCallback type="ccr" function="camModelColider"/>
    <ScriptCallback type="cmb" function="modelModelColider"/>
    <ScriptCallback type="cmd" function="modelModelColiderDuring"/>
    <ScriptCallback type="cmf" function="modelModelColiderFinish"/>
    <ScriptCallback type="generic" function="genericCallBack" />
```

```
27      <ScriptCallback type="sh" function="shaderCallback"/>
28      <ScriptCallback type="ef" function=" effectCallback "/>
29      <ScriptCallback type="timer" function="timerCallback"/>
30
31      <Effect name="zoom" effectFile ="Includes/Shaders/zoomer2.fx"/>
32
33      <GeometricElement name="transFormNodePlane"
34          type=" static "
35          parent=""
36           gfxCollision =" false "
37           collisionDist ="30"
38          pos="2900.0 0.0 0"
39          rot="0 0 0"
40           scale ="1 1 1"
41           visible ="true"
42          enableText =" false "
43           text =" light "
44          textColor="200 200 200 1"
45           size ="20 20"
46          castShadow="true">
47          <Texture materialType="s"
48              enableTextureColorKey=" false "
49              textureColorKey="20 20 20 1"
50              normalMapMultiplier="30"
51              parallaxMapHeight="20"
52              vertexAlphaColor="-1"
53              enableEmissiveColor=" false "
54              emissiveColor="20 20 20 2"
55              enableFog=" false ">
56          </Texture>
57      </GeometricElement>
58
59      <GeometricElement name="earth_kegel"
60              type=" static "
61              parent=""
62               construct =" false "
63               gfxCollision =" false "
64                collisionDist ="30"
65               file =" Includes/Models/globe_half.ms3d"
66              zipArchive=""
67              animation="0"
68              pos="0.0 0.0 0.0"
69               rot="0 180 120"
70               scale ="1 1 1"
```

```
71                visible ="true"
72               enableText=" false "
73               text=" Schriftzug  ' developers  delight ' "
74               textColor="200 200 200 200"
75              simpleShadow="false">
76              <Texture  materialType="s"
77                        enableTextureColorKey=" false "
78                        textureColorKey="0 0 0 1"
79                        normalMapMultiplier="10"
80                        parallaxMapHeight="0.002"
81                        vertexAlphaColor="-1"
82                        enableEmissiveColor="true"
83                        emissiveColor="255 255 255 1"
84                        enableFog=" false "
85                        shader=""
86                        noCulling="true"
87                        wired=" false "
88                         effect ="zoom">
89                        <TextureLayer  layer="0"  file ="RTT"/>
90              </Texture>
91            </GeometricElement>
92
93
94            <GeometricElement name="mask"
95                        type="image"
96                        pos="0.0  0.0  0.0"
97                         visible ="true"
98                        alpha="true"
99                        color="255 255 255 255"
100                       destRectangle ="-1 0 1401 1050"
101                       srcRectangle ="0 0 1400 1050"
102                       clipRectangle ="0 0 0 0">
103               <Texture  enableTextureColorKey=" false "
104                         textureColorKey="0 0 0 1">
105               <TextureLayer  layer ="0"  file =" Includes / Textures / msk.png"/>
106               </Texture>
107         </GeometricElement>
108 </Scene>
```