Carl von Ossietzky Universität Oldenburg
Projekt Gruppe 2005 / 2006

31. Mai 2006

Knut Angenendt, Kilian Asangana,
Cigdem Cebe, Matthias Davidek,
Ralf Eckert, Daniela Hans,
Florian Hinz, Inga Laßwitz,
Lars Lüttmann, Malte Mathiszig,
Christoph Meyer, Nils Peters,
Martin Schmaeck, Jing Shui,
Christian Thevissen, Christian Wellinghorst

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der Projektgruppe „3D-Entwicklungsumgebung“, die von Prof. Dr. W. Kowalk betreut wird, sollen ein Editor und eine Engine für 3D-Welten und Animationen erstellt werden. Als Rahmenbedingungen sind insbesondere die Verwendung von OpenGL und Java, bzw. Jogl[1], vorgegeben.

Im Folgenden eine Erklärung unserer Software: Dazu werden die beiden großen Komponenten *Editor* und *Engine* vorerst einzeln betrachtet, anschließend folgt die Dokumentation der *Schnittstellen* zwischen diesen Komponenten.

Der erste Abschnitt beschreibt den Editor. Zunächst wird das Model- View- Controller-Prinzip, nach dem der Editor entwickelt wurde, erklärt und auf dessen Umsetzung näher eingegangen.

Ergänzende Informationen zum Yage[2]3D-Projekt findet der interessierte Leser auf der Projektgruppen-Homepage:

http://www.yage3d.de

[1] Java Bindings for OpenGL

2 Editor

Die Klasse `Main` liegt direkt im Paket `yage3d.editor` und enthält als einzige Methode diejenige, die für den Start des Editors und damit auch unserer Anwendung zuständig ist. Sämtliche weiteren Klassen sind in Unter-Paketen nach dem Model-View-Controller-Prinzip strukturiert.

2.1 Model-View-Controller-Prinzip

Unser Editor wurde auf dem Model-View-Controller (MVC)-Prinzip aufgebaut. Dieses Prinzip, 1978 bei Xerox entwickelt [7], wird in vielen Bereichen der Softwareentwicklung eingesetzt. MVC bedeutet strikte Trennung und Aufgabenverteilung bei einer Anwendung. Datenquellen werden dabei als „Model“ bezeichnet, wobei die Daten unabhängig vom Erscheinungsbild geliefert werden. Diese Daten werden in der „View“ entsprechend angezeigt, ohne dabei vom Anwendungskern [7], dem „Model“, beeinflusst zu werden. Der „Controller“ ist für die Interaktion zwischen dem Benutzer und der Anwendung zuständig. Werden Werte vom Benutzer in der View geändert, interpretiert der Controller die Eingabe und teilt dem Model diese Änderung mit [7]. Demnach kann der Contoller auch als Logik der Anwendung bezeichnet werden [5].

Diese Aufteilung birgt viele Vorteile. Die Aufgaben können aufgeteilt werden und allgemein sollte es möglich sein, die einzelnen Komponenten auszutauschen [5].

Performancevorteile können erreicht werden, wenn nur der veränderte Bereich benachrichtigt wird und nicht die komplette Model-Schicht. [5]

Die **View** definiert, zusammen mit dem Controller, das Verhalten bei Interaktionen sowie das Erscheinungsbild der Anwendung. Beides sind demnach Komponenten der grafischen Oberfläche, die eine Schnittstelle zur GUI-Bibliothek (GUI = Graphical User Interface) und somit auch indirekt zum Benutzer bereitstellen. Die View kennt ihr Model und auch den Controller, so dass Informationen über den Zustand des Models abgerufen werden können. Die View kann alle Aspekte oder nur Teilaspekte des Model-Zustands darstellen. Sie verfügt über GUI-Komponenten, die dem Benutzer den Model-Zustand sichtbar machen und diese durch diverse Eingabemöglichkeiten verändern kann [7].

Das **Model** ist unabhängig vom Controller und der View, aus diesem Grund sollten weder Controller- noch View-Elemente im Model vorhanden sein. Die interne Datenverarbeitung ist also gänzlich abgekoppelt von der GUI, so dass die Änderungen in der GUI keine Auswirkungen auf die interne Datenverarbeitung und Datenstrukturen haben [6]. Die Model-Schicht repräsentiert die Anwendung und stellt Operationen zur Verfügung,

die vom Controller genutzt werden können, um Ereignisse umzusetzen [7]. Sie ist demnach die Schicht, die für die interne Datenverarbeitung verwendet wird [6]. Zudem werden Informationen über den Zustand der Anwendung bereitgestellt, die von der View abgerufen werden können. Ebenso lässt diese Schicht `ActionListener` zu, die bei Zustandsveränderungen aufgerufen werden, damit die View die Darstellung der Daten aktualisieren kann, bzw. eine Änderung der Darstellung durch den Controller ausgelöst werden kann. Anwendungsoperationen werden ebenfalls in dieser Schicht implementiert, das heißt, das Model besteht nicht nur aus reinen Datenklassen, sondern stellt auch Methoden zur Manipulation bereit [7].

Der **Controller** überwacht die Eingaben des Benutzers und interpretiert die Ereignisse, die von der View gesendet werden, wodurch das Model geändert werden kann [6]. Es ist sinnvoll, Anwendungsoperationen nicht im Controller zu implementieren, sondern diese Funktionen in die Model-Schicht auszulagern. Dadurch wird gewährleistet, dass nur Ereignisse der GUI vom Controller abgefangen werden und er die Änderungen an die entsprechenden Funktionen der Model-Schicht weiterleiten kann. Er definiert, welches Ereignis welche Operation auslöst [7].

2.2 Model

Die Model-Schicht unserer Anwendung realisiert sämltiche Datenstrukturen, die für die Arbeit nötig sind. Hier werden die Operationen bereitgestellt, die zur Manipulation der Datenstrukturen gebraucht werden. Die Model-Schicht könnte auch für eine andere View-Schicht, als wir sie implementiert haben, verwendet werden, da keinerlei View-Elemente in dieser Schicht realisiert wurden. Die Manipulationsmethoden dieser Model-Schicht werden in der Controller-Schicht ausgeführt, um die View dem Model anzupassen und umgekehrt. Veränderungen des Models in der View werden über die Controller-Schicht an das Model weitergeleitet und ausgeführt.

Unsere Model-Schicht ist in weitere Teilbereiche gegliedert. In einem Teilbereich werden System- und Umgebungsvariablen sowie Metadaten gespeichert und verwaltet. Ein weiterer Teilbereich kümmert sich um die Erzeugung der Objekte in der Welt. Dabei handelt es sich um Geometrie-Objekte, inklusive Materialien, Texturen und Surfaces, sowie Licht- und Trigger-Objekte. Eine weitere Schicht bietet Werkzeuge zur Manipulation an. Über diesen verschiedenen Schichten befindet sich die Klasse `MainFactory.java`, welche für die Erzeugung jeglicher Objekte zuständig ist.

2.2.1 Laufzeit

Für die Speicherung und Verwaltung der Variablen und Metadaten sind die Pakete

- `yage3d.editor.model.environment`
- `yage3d.editor.model.metadata`

zuständig.

Wie die Paket-Namen bereits vermuten lassen, bietet das Paket „environment“ Variablen und Funktionen zum Speichern und Verwalten der Editor-Umgebung während der Laufzeit an. Mit dem Paket „metadata“ werden die Metadaten der Objekte und des Projekts selbst verwaltet.

Umgebung (environment)

Das Paket beinhaltet verschiedene Klassen für unterschiedliche Aufgaben. Diese sind:

- Speicherung der Editorumgebung (Layout, Projekte)
- Einstellungsmöglichkeiten
- Das Laden und Speichern der Umgebungsvariablen, bzw. der Welt und des Projekts

Die Editorumgebung umfasst die Layout-Einstellungen der GUI, das heißt welche Komponenten wo platziert und angezeigt werden. Diese Komponenten sind die 4 Fenster für die Arbeitsfläche, die verschiedenen Werkzeugleisten (Toolbars), sowie die verschiedenen Panels. Zu den Panels gehören u.a. der Szenenbaum, der Projektbrowser und die Eigenschaftenfenster.

Ebenso werden in der Editorumgebung benutzerdefinierte Einstellungen (Preferences) verwaltet. Dazu gehören Einstellungen, welche die Darstellung im Editor manipulieren, beispielsweise Farbeinstellungen und Linienstärken. Auch die aktuelle Spracheinstellung für den Editor (englisch oder deutsch) sowie die gewünschten Einheiten (Meter, Meilen, etc.) werden hier verwaltet.

Speicherstruktur

Wir verwendeten zunächst die in Java enthaltenen DOM-Funktionen zum Laden und Speichern von Projektdaten und Welten. Beim Anlegen eines Projekts wird eine Verzeichnisstruktur angelegt, die Unterordner für Texturen, Sounds, gespeicherte Welten, Skripte und Objekte vorsieht.

Weil sich laufend das Objektkonzept des Editors änderte, mussten die Speicher- und Ladefunktionen häufig angepasst werden. Da das Parsen der komplexer werdenden Objekte mit DOM verhältnismäßig lange dauerte, entschieden wir uns später dazu, Java's SAX-Parser zu verwenden. Leider erwies sich dieser Schritt im Nachhinein als Fehler, denn die Handhabung dieses Parsers lässt doch arg zu wünschen übrig und ein Performance-Gewinn war kaum sichtbar. Letztendlich führte dies zu eine Mischform aus XML und Binärdateien im Projekt: Alle größeren Datenmengen, wie z.B. Vertices und Dreiecke werden binär gespeichert, während alle anderen Daten nach wie vor in XML-Dateien vorliegen.

2.2.2 Objekte in der Welt

Objekte, die sich in der Welt bzw. in einer Szene befinden, können Licht-, Trigger- und Geometrie-Objekte sein. Diese Objekte werden in den Szenenbaumen über spezielle Knoten (Abschnitt 2.3.7) eingehängt und so hierarchisch geordnet und strukturiert.

Geometrie-Objekte (geoobjects)

Das Paket `yage3d.editor.model.geoobject` enthält Klassen für verschiedene Aufgabengebiete:

- die Objekte an sich (statisch und animiert)
- CSG (Boolesche Operationen auf Objekten)
- Hilfsklassen, hauptsächlich zur Erstellung von Geometrie-Objekten

Objekt-Konzept

Es gibt zwei verschiedene Arten der Geometrie-Objekte, zum einen die statischen, zum anderen die animierten Geo-Objekte, wobei als Animationsverfahren Keyframe-Animation und Objekt-Hierarchie-Animation (OH-Animation) umgesetzt wurden.

statische Objekte

Grundklasse der statischen Geo-Objekte ist die Klasse `ExtendedStaticGeoObject`, welche die abstrakten Methoden von der Klasse `BaseGeoObject` aus der Engine erbt. Weiterhin implementiert sie die abstrakten Methoden der Klasse `ExtendedBaseGeoObject`.

Durch die Implementierung der Klasse `ExtendedBaseGeoObject` ist jedes Editor-Geometrie-Objekt gezwungen, eine Zeichenmethode anzubieten, damit die Geometrie-Objekte in der Lage sind, sich selbst zu zeichnen.

Jedes Geometrie-Objekt besteht aus einer oder mehreren Oberflächen, den sogenannten Surfaces, welche wiederum eine Liste von Dreiecken darstellen. Es gibt verschiedene Arten von Surfaces sowie Funktionen zur Manipulation ihrer Geometrie (Abschnitt 2.2.2).

Darüberhinaus gibt es verschiedene Grundtypen von Geometrieobjekten (z.B. Würfel, Kugeln, Kegel, ...). Die Information um welchen Typ von Objekt es sich handelt wird dabei durch Metadaten abgebildet(Abschnitt 2.1.1).

animierte Objekte

Animierte Objekte enthalten, unabhängig vom Typ der Animation, eine Liste von sogenannten Keyframes sowie eine Liste von Animationen. Die Keyframes dienen dabei

dazu, Schlüsselpositionen eines animierten Objekts zu repräsentieren zwischen denen später, um eine flüssige Bewegung zu simulieren, interpoliert werden kann. Die in den Keyframes enthaltenen Informationen variieren je nach Animationstyp. Die Animationen dienen dazu die Keyframes zu gruppieren, um so verschiedene Bewegungen (z.B. laufe, springen etc.) logisch abzubilden.

Wie bereits erwähnt, wurden von uns zwei verschiedene Animationsverfahren implementiert.

Ein **OH-animiertes** Geo-Objekt ist eine Art Container, der weitere, hierarchisch als Baum geordnete, Geo-Objekte enthält. Die Keyframes eines OH-animierten Objekts enthalten lediglich Informationen über die einem enthaltenen Geometrieobjekt zugeordneten Verschiebungs- bzw. Rotationsfaktoren genenüber dem ihm übergeordneten Objekt. Für mehr Informationen zum Thema OH-Animation siehe Abschnitt 3.2.1.

Ein **keyframe-animiertes** Geo-Objekt wird, ebenso wie die statischen Objekte, durch die Klasse `ExtendedStaticGeoObject` abgebildet. Dies ist deshalb sinnvoll möglich, da bei einem keyframe-animierten Objekt in jedem einzelnen Keyframe die gesamte Objekt-Geometrie in der jeweiligen Position gespeichert wird. Daher kann ein einzelner Keyframe sowohl die Geometrie eines statischen Objekts abbilden, als auch eine einzelne Schlüsselposition eines keyframe-animierten Objekts. Für mehr Informationen zum Thema Keyframe-Animation siehe Abschnitt 3.2.1.

CSG

CSG bedeutet „Constructive Solid Geometry“. CSG ist in der Lage, eine prinzipiell unbegrenzte Anzahl konvexer 3D-Objekte miteinander zu kombinieren. Bei unserer Implementation sind Addition und Subtraktion möglich, deshalb spielt die Reihenfolge der Operationen eine entscheidende Rolle, da ein Objekt, das subtrahiert wird, unterschiedliche Auswirkungen auf die Objekte hat, die davor hinzugefügt wurden bzw. danach addiert werden.

Alle Polygone jedes Objekts werden gegen die Schnittebenen, die durch die Polygone aller anderen Objekte definiert werden, klassifiziert, das heißt es wird festgestellt, ob die Polygone vor, hinter oder auf dieser Ebene liegen. Nach dieser Klassifikation werden die Polygone entsprechend des Ergebnisses behandelt (d.h. ggf. geschnitten oder auch in mehrere Polygone unterteilt) und dem resultierenden Objekt hinzugefügt. Bei dieser Klassifikation werden verschiedene Rahmenbedingungen unterschieden und berücksichtigt (z.B. ob das Objekt addiert oder subtrahiert wird und ob das Objekt gegen das es gerade geschnitten wird in der Objektsequenz vor oder nach dem gerade Behandelten liegt). Beim Schneiden eines Polygons wird nicht nur die Geometrie der ursprünglichen Einzelobjekte verändert, sondern es werden auch die Texturkoordinaten neu berechnet. Am Ende werden schließlich noch mehrfach vorhandene Vertices, die während des Prozesses erzeugt wurden, entfernt.

Einem CSG-Objekt können zu jedem Zeitpunkt neue Objekte hinzugefügt werden - wichtig ist dabei, daß alle Grundobjekte aus denen es zusammengesetzt wurde noch bekannt sind, da bei jeder Neugenerierung der Geometrie die ursprünglichen, konvexen Objekte benötigt

werden.

Surface-Konzept

Geometrieobjekte bestehen prinzipiell aus einer Menge von Dreiecken (Faces) die über einer Menge von Vertices aufgespannt sind. Diese Dreiecksmenge kann bei einigen Geometrieobjekten (beispielsweise bei Kugeln) derart groß werden, dass es unpraktikabel wäre, bei Manipulationen der Geometrie jeden Vertex einzelnd per Hand bearbeiten zu müssen. Daher werden die Dreiecke der Geometrieobjekte bereits bei ihrem Erstellen zu logischen Gruppen zusammengeschlossen, die durch anschauliche einfache Oberflächen, den so genannten Surfaces, repräsentiert werden. Zusätzlich verfügt jedes Surface intern über eine Abbildung von 2D-Koordinaten auf die beteiligten Vertices, welche u. A. als Grundlage für Texturierungen und Oberflächenmanipulationen dient.

Durch Surfaces ist es möglich, durch Manipulation einiger weniger geometriebestimmender Vertices (so genannter MainVertices) ein Surface komfortabel zu bearbeiten. Weitere Vertices, die zur feineren Unterteilung des Surfaces dienen (so genannte SubVertices), werden dann anhand der Lage der MainVertices interpoliert, ohne die Struktur des Surfaces zu zerstören.

Auf diese Weise ist es sogar möglich, die Feinheit der Unterteilungen eines Surfaces, also die Anzahl der Dreiecke, aus denen es besteht, unabhängig von dessen Lage und Form auch nachträglich zu verändern. Ebenso ist ein beliebiger Wechsel zwischen runden oder kantigen Normalen[1] möglich.

Durch die interne Kenntnis der 2D-Koordinaten der Vertices, ist es ein Leichtes, ein Surface mit Colormaps oder Texturen zu versehen. Darüber hinaus besteht die Möglichkeit, Surfaces durch so genannte Surfacetransformationen (siehe dort), bei denen es sich um Displacement Maps handelt, zu verformen. Auch hier gilt, dass durch die intelligente interpolierte automatische Berechnung der Vertices, Surfacestransformationen zu einem beliebigen Zeitpunkt hinzugefügt oder wieder gelöscht werden können.

Selbstverständlich ist es auch möglich, ein Geometrieobjekt in ein sogenanntes FreeMesh zu konvertieren, bei dem jeder Vertex einzelnd manipulierbar ist. Selbst derartige FreeMesh-Geometrieobjekte lassen sich noch beliebig durch Surfacetransformationen manipulieren.

Alle Surfaces sind von der Oberklasse `MotherSurface` abgeleitet. Es gibt die folgenden Surfacetypen, aus denen Geometrieobjekte bestehen können:

- FreeSurface (für vertexgenaue Manipulationen, CSG und Objekte aus dem Polygoneditor)
- ConeSurface (für Kegel)
- DiscSurface (für Scheiben, Kegel- und Zylinder-Deckel)
- QuadSurface (für Rechtecke und Quader)
- CirculationSurface (für Zylinder)

[1] Eigentlich sind Normalen senkrecht auf einer Fläche stehende Vektoren (kantige Normalen). Bei runden Normalen handelt es sich um eine Technik, durch die beleuchtete Kanten von Geometrieobjekten weicher, also runder, erscheinen.

- SphereSurface (für Kugeln)
- TriangleSurface (für Pyramiden)

Licht-Objekte

Ein Licht-Objekt aus dem Paket `yage3d.editor.model.lightobject` repräsentiert eine Lichtquelle in der Welt. OpenGL ist in der Lage, maximal 8 dynamische Lichtquellen gleichzeitig darzustellen. Diese Beschränkung wurde in diesem Licht-Objekt berücksichtig. Die einzelnen Lichtquellen werden, je nach Art, als gelber Kegel (Spotlight) oder als gelbe Kugel im Editor dargestellt, um die Bearbeitung der Lichtquellen zu vereinfachen. Mehr zum Thema „Licht“ gibt es im Abschnitt 3.2.1

Trigger-Objekte

Bei einem Trigger-Objekt handelt es sich im Wesentlichen um ein Bounding Volume, dem kein Objekt mit sichtbarer Geometrie zugeordnet ist. Trigger-Objekte dienen dazu Bereiche in der Welt zu definieren, bei deren Betreten / Durchqueren / Verlassen durch den Spieler (oder allgemein: eines bewegbaren Objektes) Ereignisse ausgelöst werden.

Trigger-Objekte unterstützen drei verschiedene Ereignisse:

- onEnter: dieses Ereignis wird erzeugt, wenn der Spieler den Bereich betritt
- onLeave: dieses Ereignis wird erzeugt, wenn der Spieler den Bereich verlässt
- onStay: dieses Ereignis wird solange (in jedem Tic) wiederholt erzeugt, wie sich der Spieler im Bereich aufhält.

Für jedes dieser drei Ereignisse kann vom Benutzer Jave-Quellcode angegeben werden, der die Reaktion auf das Ereignis beschreibt. Neben dem Umfang der Sprache Java steht dem Anwender insbesondere die Klasse YageAPI zur Verfügung, die diverse Methoden zur Laufzeit-Manipulation der Welt im Runtime-Modul bereitstellt. Beispielsweise lassen sich über die YageAPI Geometrieobjekt-Animationen steuern, Lichter ein- und ausschalten oder Sounddateien abspielen.

Aus diesen und weiteren Code-Fragmenten wird in der Compilierungsphase automatisch eine Java-Datei erstellt, die die Spiel- und Steuerlogik des RuntimeModuls enthält. Diese Java-Datei kann sogar zur Editorlaufzeit kompiliert (falls ein Javacompiler installiert ist) und durch einen eigenen ClassLoader in die Vorschau des RuntimeModuls geladen werden: So läßt sich die Welt samt gerade verändertem Code im Runtimemodul direkt aus dem Editor heraus testen.

2.2.3 Aussehen der Geometrie-Objekte

Geometrie-Objekte können durch verschiedene Methoden zu einem neuen Aussehen kommen. Damit ist keine neue Basis-Struktur der Geometrie an sich gemeint, sondern das Aussehen der Oberflächen des Geometrie-Objekts.

Geometrie-Objekte können mit einer oder mehreren Farben versehen werden. Dieses ist die einfachste Möglichkeit, einem Objekt ein anderes Aussehen zu verleihen. Weiterhin können Materialien auf das Geo-Objekt gelegt werden. Materialien können nur einfache Texturen sein, aber sie können auch mit weiteren Eigenschaften versehen werden, bis hin zum Multitexturing mit Farbverläufen.

Wie bereits oben beschrieben gibt es auch die Möglichkeit, die Oberflächen der Geo-Objekte durch eine mathematische Funktion zu transformieren, um dem Geometrie-Objekt ein verändertes Aussehen zu verleihen.

Farbgebung

Man kann die Geo-Objekte mit einer und mit mehreren Farben versehen. Bei einem Würfel zum Beispiel können alle sechs Seiten eine unterschiedliche Farbe haben. Darüber hinaus kann eine Fläche auch mit mehreren Farben ausgestattet werden. Dazu wird jedem Eckpunkt eine Farbe zugewiesen, welche dann zur Mitte hin verläuft. So kann man durch Farbverläufe schöne Effekte erzielen.

Das Objekt oder die Eckpunkte (Vertices) des Objekts können mit der Maus ausgewählt werden (Picking), in einem Eigenschaftenfenster kann über dann einen Farbwahl-Dialog die Farbe ausgewählt werden, die dem Objekt oder dem Eckpunkt (Vertex) zugewiesen werden soll.

Texturen und Material

Eine Textur ist eine Bild-Datei, die auf eine Oberfläche des Geometrie-Objekts gelegt wird. Diese Texturen können im Texturbrowser ausgewählt werden. Es ist notwendig, Bild-Dateien aus anderen Verzeichnissen zuerst in das Projekt und anschließend in den Texturbrowser zu importieren. Dieser bietet die nötige Funktionalität an. Wenn man im Datei-Browser Bild-Dateien anwählt, wird im rechten Teil des Browsers eine Vorschau des Bildes angezeigt. So wird dem Nutzer die Textur-Auswahl erleichtert.

Nachdem sich die Texturen im Textur-Ordner des Projekts befinden, werden diese im Texturbrowser angezeigt. Nun kann eine Textur auf das Geo-Objekt gelegt werden, indem man das gewünschte Surface des Objektes auswählt und auf den Button neben Material klickt.

Hier kann man dann Texturen in einem Dialog aus dem Browser auswählen und für das Surface zusammenstellen. Mit einem Klick auf „Preview“ kann man das Ergebnis betrachten, und, wenn es für gut befunden wird, die Einstellungen übernehmen.

Material (material)

Jedes Surface besitzt ein eigenes Material. Dieses kann aus einem einfachen Farb-Mapping oder aus einer Textur bestehen, es kann aber auch mehrfach texturiert und mit Farbverläufen versehen sein.

Die wichtigste Klasse in diesem Zusammenhang ist die `yage3d.engine.objects.materialobjects.Material.java`. Sie übernimmt die Generierung und die Verwaltung der Materialien. Es werden dazu in dem Paket `yage3d.editor.model.material` verschiedene Mapping-Möglichkeiten zur Verfügung gestellt. Diese für das Mapping zuständigen Klassen werden von der Klasse `TextureMapping` abgeleitet, welche die Methoden zur Berechnung der Texturkoordinaten zur Verfügung stellt. Folgende Mapping-Möglichkeiten sind vorgesehen:

- Farb-Mapping
- normales Textur-Mapping
- Mapping mit Hilfe von Jython Skripten

In diesem Paket liegen auch die für die oben beschriebene Farbgebung verwendeten Klassen. Dabei ist `ColorMapping` die Oberklasse für alle farbgebenden Mappings.

Surfacestransformationen: Displacement Maps

Bei einer Surfacetransformation handelt es sich um eine geometrieverändernde Displacement Map, also um eine texturartige Karte, die bei einem Surface die Lage der Vertices in Normalenrichtung[2] verschieben kann. In Yage wird zwischen drei Arten von Surfacetransformationen unterschieden:

- SurfaceTransformationFileDisplacementmap: Basiert auf einer Höhenkarte aus einer Bilddatei.
- SurfaceTransformationRandomDisplacementmap: Erstellt eine parametergesteuerte Zufallskarte.
- SurfaceTransformationJython: Erstellt eine mittels Jython geskriptete Höhenkarte.

Bei Höhenkarten aus Bilddateien werden Graustufen-Bilder vorausgesetzt, deren Helligkeitswert als Höhenwert interpretiert wird.

Das Besondere an der Jython-basierten Surfacetransformation ist, dass zur Berechnung des Höhenwertes nicht nur die 2D-Koordinaten (a, b) als Parameter dienen, sondern dass die Berechnung zusätzlich von der Keyframe-Nummer abhängig ist. Damit lassen sich auf einfache Weise Keyframe-Animationen mit Oberflächen-Animationen (beispielsweise

[2] Normalenrichtung = aus Sicht der Oberfläche nach oben.

Wellen) versehen. Ausserdem können Jython-Skripte bei ihrer Berechnung des jeweiligen Höhenwertes auf die Berechnung anderer Surfacestransformationen zugreifen und natürlich auch die reichhaltigen Funktionen der Java-Math-API nutzen.

Da bei Surfacetransformationen (Displacement Maps) tatsächlich die Geometrie verändert wird, ist es natürlich erforderlich, dass das betreffende Surface über eine ausreichend hohe Kacheldichte verfügt, damit ausreichend viele verschiebbare Vertices verfügar sind.

2.2.4 Werkzeuge

Die Pakete `yage3d.editor.model.system` und `yage3d.editor.model.tools` bieten Werkzeuge zur Manipulation und zur Verwaltung der Daten an, wobei die Verwaltung der Daten im Paket „System“ realisiert wird und die Unterstützung der Manipulation von „Tools“ gewährleistet wird.

Verwaltung (System)

Das Paket `yage3d.editor.model.system` beinhaltet die Objektverwaltung und es bietet verschiedene Filter für unterschiedliche Dateien an.

Objektverwaltung

Objekte können mit Hilfe dieses Teilbereichs gruppiert und strukturiert werden. Wenn mehrere Objekte gewählt werden, können diese gruppiert und als Gruppe weiter manipuliert werden. Dabei wirkt sich die Manipulation auf alle Objekte der Gruppe gleich aus. Wird eine Gruppe verschoben, verschieben sich sämtliche, zu der Gruppe gehörenden Objekte. Natürlich kann die Gruppierung auch wieder rückgängig gemacht werden.

Durch einen `HideController` können Objekte oder auch ganze Objekt-Gruppen ausgeblendet werden. Sie werden dann nicht mehr in der aktuellen Szene gezeichnet. Dieses fördert die Modellierung der Welt, da man ganze Abschnitte ausblenden kann, um an einzelnen Objekten in Ruhe arbeiten zu können, ohne dabei von gerade nicht benötigten Objekten gestört zu werden.

Zur Objektverwaltung gehört ebenfalls eine Zwischenablage. In die Zwischenablage werden kopierte und ausgeschnitte Objekte gelegt, die wieder in die Welt eingefügt werden können. Sie wird automatisch angeregt durch die gängigen Short Cuts „strg + c“ für Kopieren, „strg + x“ für Ausschneiden und „strg + v“ für Einfügen, sowie durch Auswahl der jeweiligen Menü-Einträge im „Bearbeiten“-Menü.

Manipulation

Das Paket `yage3d.editor.model.tools` stellt ein Selektionsfeld und das im Abschnitt 2.2.2 genannte Tesselation zur Verfügung.

Das Selektionsfeld dient zur Mehrfachauswahl. Es kann mit der Maus um verschiedene Objekte gezogen werden, welche dadurch selektiert sind und weiter bearbeitet werden können.

Tesselation wird, wie schon erwähnt, zur Verfeinerung der Gitterstruktur der Geometrie-Objekte verwendet. Ebenso ist es möglich, durch Tesselation neue Vertices dem Objekt hinzuzufügen, welche dann automatisch in die Gitterstruktur einbezogen werden.

2.3 View

Die View-Schicht unserer Anwendung definiert das Layout des Editors. Das Verhalten bei Interaktionen ist teilweise über diverse Listener in die Controller-Schicht ausgelagert, jedoch werden hier auch Dialoge zur Benutzer-Interaktion zur Verfügung gestellt. Die View-Schicht unserer Anwendung beinhaltet die gesamte GUI (Graphical User Interface). Wir haben uns dabei für Komponenten aus dem Paket `javax.swing` entschieden, da das Arbeiten mit diesen Komponenten komfortabler ist als mit Komponenten aus `java.awt`.

Die View-Schicht ist an das Model gekoppelt, da zum Beispiel über die Werkzeugleisten Datenstrukturen aus dem Model in den Editor geladen werden können. Somit ist auch der Zustand der Model Schicht ganz oder auch nur teilweise in der View abbildbar. Durch Dialoge und Listener kann der Benutzer das Model in der View verändern.

In diesem Abschnitt werden die Teilbereiche der GUI näher beschrieben. Einige GUI-Komponenten unterscheiden sich in den verschiedenen Editor-Modi „Normal“ und „OH-Modus“. Unterschiedlich sind hier die Werkzeugleisten und die Eigenschaftenfenster sowie die Browser.

Die folgenden Komponenten, Hilfe, Ansichten und Konfiguration, benutzen beide Modi, weshalb eine Trennung sinnfrei wäre. Auf die Unterschiede wird in den folgenden Abschnitten eingegangen.

2.3.1 Start

Wenn der Editor gestartet wird, erscheint ein Ladebildschirm mit unserem Logo und

Abbildung 2.1: Start-Fenster

einem Status-Balken (Abbildung 2.1). Anschließend öffnet sich der `StartDialog`, in dem man auswählen kann, ob ein bereits vorhandenes Projekt geöffnet oder ein neues Projekt erstellt werden soll (Abbildung 2.2).

Wird **„neues Projekt anlegen“** gewählt, öffnet sich ein Fenster (Abbildung 2.3)

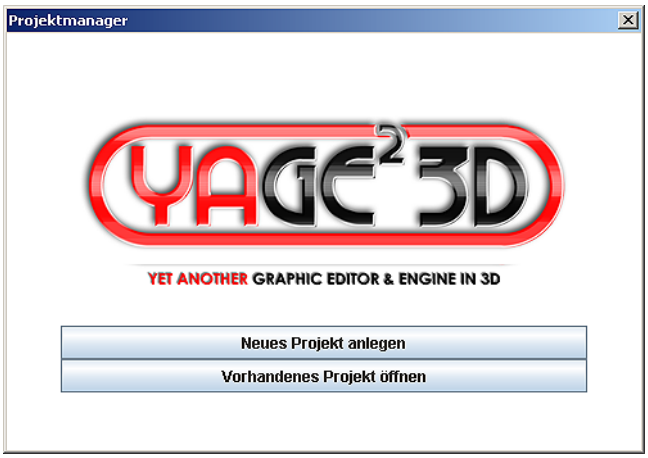


Abbildung 2.2: `StartDialog`

mit einer Ordner-Auswahl und einem Textfeld, in dem der Name des neuen Projekts eingetragen werden kann. Als default-Ordner wird der Pfad zum „Eigene Dateien“-Ordner (Windows) bzw. das Home-Directory (Linux) angegeben. Der Benutzer kann diesen Pfad mit dem „Ändern“-Button anpassen.

Beim Klicken des Buttons öffnet sich ein FileChooser, der nur Verzeichnisse wählen kann.

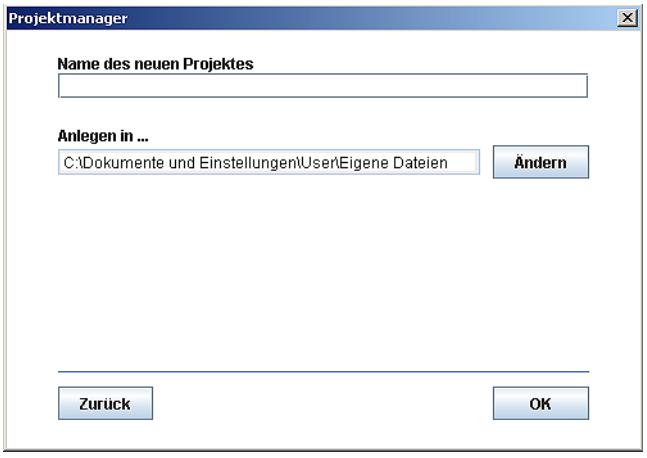


Abbildung 2.3: Projekt anlegen

Wenn der gewünschte Pfad und der Name eingegeben ist, wird durch den „OK“-Button ein neues Fenster geöffnet. Es gibt ebenfalls die Möglichkeit, einen Schritt zurück zu gehen, falls man sich umentschieden hat. In dem neuen Fenster (Abbildung 2.4) muss nun eine neue Welt angelegt werden, um mit dem Editor arbeiten zu können. Per default wird

eine Welt mit dem Namen „world-1“ angelegt. In diesem Fenster ist es ebenfalls möglich, vorhandene Welten zu löschen oder das Projekt zu schließen. Wenn man eine neue Welt angelegt hat, so kann man sie mit dem „Welt öffnen“-Button unten rechts öffnen und mit der Modellierung beginnen.

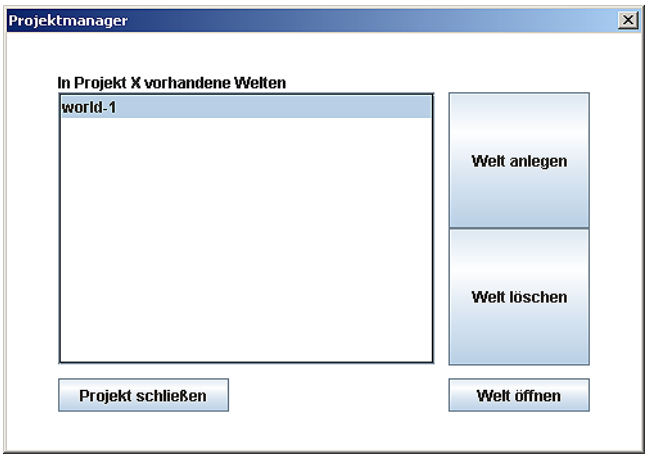


Abbildung 2.4: `WorldBrowser`

Wird **„vorhandenes Projekt öffnen“** gewählt, so wird standardmäßig das „Eigene Dateien“-Verzeichnis (Windows) oder das Home-Directory (Linux) geöffnet. In dem FileChooser kann nun das Verzeichnis des Projekts gewählt und die `project.xml` geöffnet werden. Anschließend öffnet sich das Fenster mit den vorhandenen Welten (Abbildung 2.4), in dem auch hier neue Welten angelegt oder vorhandene Welten gelöscht werden können. Nachdem eine Welt gewählt wurde, kann man nun diese öffnen und mit der Modellierung fortfahren.

Schließt man das Start-Fenster mit dem „`x`“ oben rechts in der Titelleiste, so öffnet sich zwar der Editor, allerdings sind alle Buttons und Menü-Einträge inaktiv, außer die Einträge „Neu“, „Öffnen“ und „Beenden“ im Hauptmenü unter „Projekt“, der Hauptmenü-Eintrag „Hilfe“, sowie die Buttons „Neu“ und „Speichern“ in der Editor-Werkzeugleiste.

2.3.2 Ansicht

Das Editorfenster (Abbildung 2.5), realisiert durch die Klasse `MainFrame`, lässt sich grob in drei Bereiche einteilen. Im oberen Bereich befindet sich die Hauptmenüleiste und darunter das Panel mit den variablen Werkzeugleisten (1). Am rechten Rand ist ein Panel, der `ObjectFrame`, in dem die verschiedenen Browser und die Eigenschaftenfenster angezeigt werden (3). Den größten Bereich nimmt der `WorkFrame` ein, welcher die verschiedenen Arbeitsansichten anzeigt (2). Dieser Bereich ist von dem `ObjectFrame` getrennt. Der `ObjectFrame` kann ausgeblendet werden und die Größe des ObjectFrames kann angepasst werden, jedoch beträgt die minimale Größe etwa 1/3 des Bildschirms.

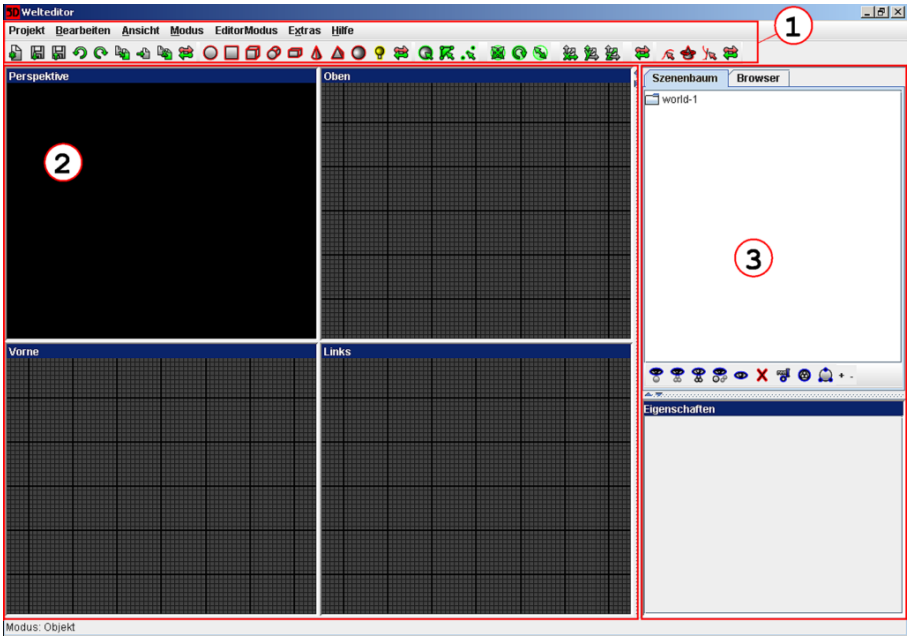


Abbildung 2.5: Editor

Arbeitsfläche

Die Arbeitsfläche zeigt per default die vier verschiedenen Arbeits-Ansichten, wobei jede der Ansichten gleich groß dargestellt wird (Abbildung 2.6). Es ist auch möglich, jeweils nur eine Ansicht auf Größe des WorkFrames anzeigen zu lassen. Ebenso lässt sich die gewünschte Ansicht aus dem Editor lösen und ist dann als Vollbild anzeigbar.

In der Perspektiv-Ansicht werden die Objekte als 3D-Körper dargestellt, mitsamt Materialien und Lichteffekten. In den anderen Ansichten wird dahingegen nur ein 2D-Drahtgitter-Modell von der jeweiligen Seite (oben, vorne und links) angezeigt.

Hauptmenü

Die Hauptmenü-Leiste (Abbildung 2.7) besteht aus verschiedenen Menü-Einträgen, die jeweils ein Untermenü öffnen. Im Menüpunkt **Projekt** können Datei-Operationen ausgeführt werden. Neue Projekte können erstellt, vorhandene Projekte geöffnet und das aktuelle Projekt gespeichert werden. Weiterhin kann man die Eigenschaften des Projekts ändern und den Editor beenden. Wenn man den Eintrag „Eigenschaften“ wählt, wird das Welt-Auswahlfenster (Abbildung 2.3) geöffnet.

Unter **Bearbeiten** sind gängige Operationen, namentlich „kopieren“, „ausschneiden“, „einfügen“ und „alles markieren“, auswählbar.

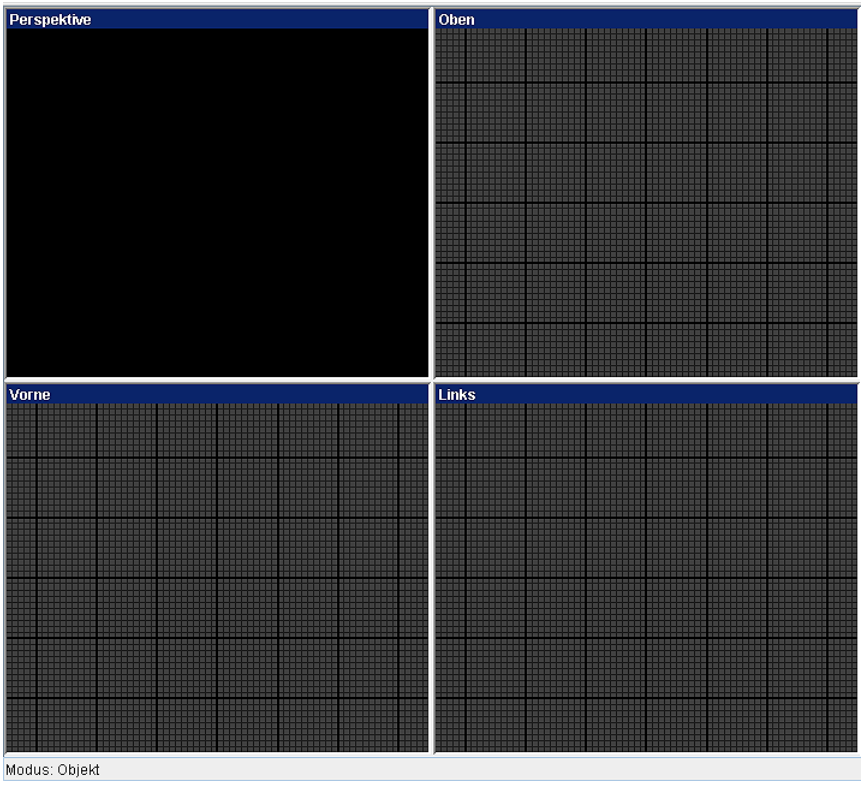


Abbildung 2.6: Arbeitsfläche

Abbildung 2.7: Hauptmenü-Leiste

Bei **Ansicht** kann der Benutzer die Arbeitsfläche und deren Aufteilung auswählen. Es können entweder alle vier Ansichten gezeigt werden, oder nur die Perspektiv-, die Front-, die Top- oder die Links-Ansicht. Ebenso kann der Benutzer die variablen Werkzeugleisten ein- und ausblenden. Dieses sind die Objekt-, Editor-, Funktions- und die Extra-Werkzeugleiste, die im Abschnitt 2.3.2 näher beschrieben werden. Weiterhin können die Browser im ObjectFrame aktualisiert, Trigger angezeigt und das Layout auf Standard-Werte zurückgesetzt werden.

Der Bearbeitungsmodus ist unter **Modus** einstellbar. Die verfügbaren Berabeitungsmodi sind „Vertex“, „Fläche“, „Objekt“ und „Freehand“. Der gewählte Modus bestimmt, welcher Picking-Modus aktiv ist. Wenn man im Vertex-Modus arbeitet, werden mit der Maus nur Vertices (Eckpunkte) des Objekts selektierbar. Analog dazu die weiteren Modi.

Der **Editor-Modus** wird im gleichnamigen Menüeintrag festgelegt. Hierbei stehen der „Normal“- und der „OH-Animationsmodus“ zur Verfügung. Dabei wird der Normal-Modus zur Generieung der Welt und der OH-Animationsmodus zum Erstellen einer Objekt-hierarchischen Animation verwendet.

Im **Extras**-Menü können globale Editoreinstellungen vorgenommen werden (Abschnitt 2.3.4). Außerdem können hier die speziellen Editoren für Jython-Skripte (Abschnitt 5.3) und Polygone (Abschnitt 2.5.1) geöffnet werden.

Die **Hilfe** kann in diesem Menü aufgerufen werden, ebenso eine Informationsseite über die Software.

Werkzeugleisten

In dieser Anwendung gibt es verschiedene Arten von Werkzeugleisten, die sich im Paket `yage3d.editor.gui.components.toolbars` befinden. Einerseits gibt es variable Werkzeugleisten, die vom Benutzer aktiviert, angepasst und verschoben werden können. Andererseits gibt es feste Werkzeugleisten, auf deren Aussehen der Benutzer keinerlei Einfluß hat. Die Icons der Buttons befinden sich für alle Werkzeugleisten im Paket `yage3d.editor.gui.data`. Jede Werkzeugleiste ist von der Oberklasse `Toolbar` abgeleitet.

Variable Werkzeugleisten

Die Werkzeugleisten unterhalb des Hauptmenüs sind variabel, das heißt, sie können ein- und ausgeblendet werden und von dem Editor-Fenster abgedockt werden. Wenn sie abgedockt sind, kann die horizontale Orientierung in eine Vertikale umgewandelt werden.

Zum Ein- bzw. Ausblenden wird das Hauptmenü verwendet. Unter **Ansicht** → **Werkzeugleisten** können die variablen Werkzeugleisten *Objekt*, *Editor*, *Funktion* und *Extra* aktiviert bzw. deaktiviert werden.

Editor-Werkzeugleiste

Die Editor-Werkzeugleiste (Abbildung 2.8) ist für typische Editor-Funktionen zuständig. Sie unterstützt ausgewählte Operationen, die auch im Hauptmenü zu finden sind. Das wären Projekt neu, Projekt öffen und Projekt speichern aus dem Menüpunkt „Projekt“ des Hauptmenüs, sowie Undo, Redo, Kopieren, Einfügen und Ausschneiden aus dem Menüpunkt „Bearbeiten“.

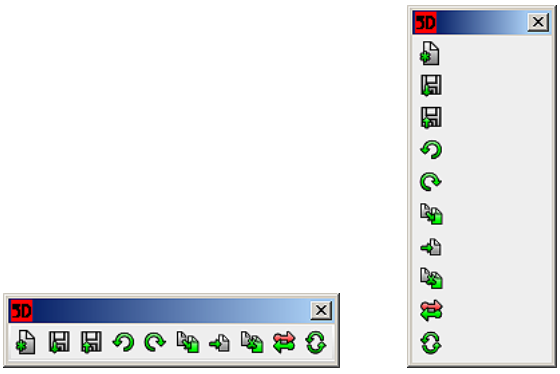
Der Button ganz rechts () dient zum Abdocken der Werkzeugleiste. Benutzt man

Abbildung 2.8: `EditorToolbar`

diesen, wird die Werkzeugleiste horizontal in einem eigenständigen Fenster dargestellt (untere Abbildung).

Dabei wird ein weiterer Button der Werkzeugleiste hinzugefügt (). Dieser verändert die Orientierung der Werkzeugleiste. Wenn sie zuvor horizontal dargestellt wurde, wird sie

nun vertikal angeordnet:



Orientierungsmöglichkeiten der Werkzeugleiste

Andersherum funktioniert es analog. Um die Werkzeugleiste wieder an das Editor-Fenster zu binden schließt man einfach das Fenster mit der Werkzeugleiste. Anschließend befindet sich diese wieder unterhalb des Hauptmenüs.

Objekt-Werkzeugleiste

Die Objekt-Werkzeugleiste hat in den verschiedenen Editor-Modi unterschiedliche Funktionalität, daher wird sie für den Normal-Modus im Abschnitt 2.3.7 sowie für den OH-Animationsmodus im Abschnitt 2.3.8 näher erläutert.

Funktion-Werkzeugleiste

Die Funktions-Werkzeugleiste (Abbildung 2.9) ist für verschiedene Operationen auf Geometrie-Objekten sowie Berabeitungsmodi zuständig. Dabei werden, wie bei der Editor-Werkzeugleiste auch, Einträge aus dem Hauptmenü mit verwendet.

Abbildung 2.9: `FunctionToolbar`

Die Auswahlmodi Objekt, Fläche und Vertex sind auch im Hauptmenü unter „Modus“ zu finden. Die weiteren Operationen besitzen jedoch keinen Eintrag im Hauptmenü. Hier werden weitere Modi angeboten, die sich auf Mausbewegungen beziehen: Bewegung, Rotation und Skalierung. Weiterhin befindet sich in dieser Werkzeugleiste auch die Achsensperrung, so dass die X-Achse, die Y-Achse und die Z-Achse separat voneinander gesperrt werden können.

Analog zur Editor-Werkzeugleiste lässt sich die Funktion-Werkzeugleiste durch an- bzw.

abdocken. Ist sie abgedockt, kann man ebenso mit die Orientierung der Werkzeugleiste im Fenster ändern.

Extras-Werkzeugleiste

Die Extra-Werkzeugleiste (Abbildung 2.10) ist für komplexe Geometrien zuständig. Durch die bereitgestellten Funktionen können Freihand-, Lathe- und Spline-Objekte erzeugt werden. Bei Freihand-Objekten öffnet sich ein neues Fenster, das den Freehand-Editor

Abbildung 2.10: `ExtraToolbar`

(x.x) repräsentiert. Lathe-Objekte sind Rotationsobjekte, die in einem extra Polygon-Editor (x.x) erstellt werden. In diesem Editor können auch die Spline-Objekte erzeugt und bearbeitet werden.

Analog zur Editor-Werkzeugleiste lässt sich die Extra-Werkzeugleiste durch an- bzw. abdocken. Ist sie abgedockt, kann man ebenso mit die Orientierung der Werkzeugleiste im Fenster ändern.

Feste Werkzeugleisten

Die festen Werkzeugleisten sind Modus-spezifisch, weshalb sie im Abschnitt 2.3.2 für den Normal-Modus und im Abschnitt 2.3.8 für den OH-Animationsmodus erläutert werden.

Objekt-Fenster

Der `ObjectFrame` ist, wie die Werkzeugleisten, in beiden Editor-Modi verschieden. Im Normalmodus sind hier der **Szenenbaum** und der **Browser** zu finden (Abschnitt 2.3.7), im OH-Animationsmodus werden hier allerdings die **Objekthierarchie**, der **Objectbrowser** und der **Animationbrowser** (Abschnitt 2.3.8) angezeigt.

2.3.3 Konfiguration

Die Gui-Konfiguration übernimmt das Paket `yage3d.editor.gui.config`. Hierbei werden Klassen zum Auslesen (`ReadConfig`) und zum Schreiben (`WriteConfig`) der Konfiguration bereitgestellt. Zur Konfiguration gehören die Spracheinstellung des Editors (`SelectLanguage` und die Position der Werkzeugleisten (`ToolbarPosition`. Zudem bietet dieses Paket eine Klasse an, mit der neue Sprachen als .jars in den Editor eingebunden und auch ausgelesen werden können (`ReadJAR`). In diesen .jar-Dateien befindet sich die Sprach-Konfigurationsdatei (.conf) sowie die Hilfe-Seiten der kontextsensitiven Hilfe. Um die Hilfe darzustellen, wird die Klasse `ExtractHTML` benötigt, die bei einer neuen Sprache die im .jar vorhandenen HTML-Seiten extrahiert.

2.3.4 Einstellungen (Preferences)

Abbildung 2.11: Einstellungen

Im Menü „Einstellungen“ können verschiedene benutzerdefinierte Einstellungen vorgenommen werden. Bei den Einstellungen handelt es sich hauptsächlich um Layout-Einstellungen, aber auch die Sprache des Editors kann angepasst werden. Folgende Bereiche sind anpassbar:

- Sprache
- Grid
- Kamera
- Orthogonal
- Selektion
- Licht
- Bounding Volumes
- 3D-Ansicht
- Objekte
- Animation
- Trigger
- Maßeinheit
- Editorlayout

Dabei gibt es für jeden Unterpunkt individuelle Einstellungen, die in den jeweiligen `Settings` abgespeichert werden. Für jeden Bereich in den Einstellungen ist eine eigene Klasse für die Speicherung zuständig.

Meistens können Farben angepasst oder Linienstärken verändert werden. Die Farbauswahl geschieht entweder über einen ColorChooser (Farbauswahlfenster, Abbildung 2.12), oder der Hexadezimalcode der gewünschten Farbe kann in ein dafür vorgesehenes Textfeld eingetragen werden. Im Farbauswahlfenster kann man zwischen drei verschiedenen Farbvorschauen wählen. Man kann entweder aus einer Palette Web-kompatibler Farben (Muster) wählen, oder, falls die gewünschte Farbe nicht dabei ist, aus HSB oder RGB-Reglern die gewünschte Farbe zusammenmischen.

Bei Linienstärken und anderen Zahlenwerten besteht die Möglichkeit, die Parameter durch up-and-down Buttons der Eigenschaftenregler, oder durch direktes Eintragen des gewünschten Wertes in das Textfeld zu verändern.

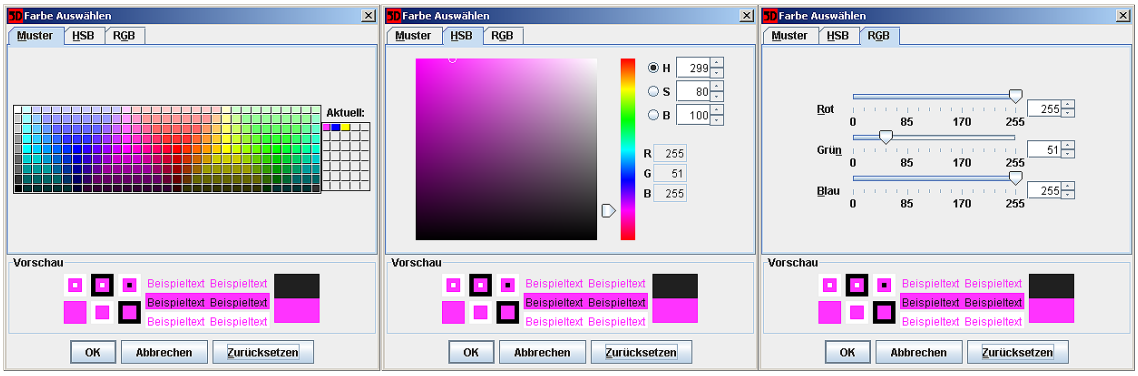


Abbildung 2.12: Farbauswahlfenster

Sprache

Bei *Sprache* kann die Edtior-Sprache angepasst werden. Als wählbare Sprachen stehen hier **deutsch** und **englisch** zur Verfügung. Der Editor passt sich sofort der ausgewählten Sprache an.

Grid

Bei *Grid* kann das Layout des Grids angepasst sowie das Grid selbst sowie Snap-to-Grid aktiviert bzw. deaktiviert werden.

Das Layout des Grids besteht aus der **Gridgröße**, die **Gridunterteilung**, der **Gridfarbe** und der **Linienstärke** des Grids. Dabei können die Haupt- und Zwischenlinien des Grids unabhängig voneinander angepasst werden. Die Gridunterteilung gibt die Anzahl der Zwischenlinien an.

Durch Snap-to-Grid werden die Positionen der Geometrie-Objekte bzw. der Vertices, je nach Auswahl-Modus, bei Bewegung auf den beim Loslassen der Maustaste nächsten Gitterknoten angepasst. Der Ursprung des Geometrie-Objekts wird dabei auf den Gitterknoten gelegt. Dieses bedeutet jedoch nicht, dass die Eckpunkte des Geometrie-Objekts notwendigerweise auf einem Gitterknoten liegen.

Kamera

Bei *Kamera* können die Rotations- und Bewegungsrichtungen (Translation) der Kamera sowie ein Rotationswert und ein Translationswert eingestellt werden.

Die Kamerabewegungen werden durch sameTrasCamDir und sameRotCamDir beschrieben. Der Wert **sameRotCamDir** gibt an, ob die Kamera mit oder entgegengesetzt zur Mausbewegung rotiert werden soll. **SameTransCamDir** steht analog für die Translationsbewegung.

Der **Translationswert** und der **Rotationswert** geben an...

Orthogonal

Orthogonal bedeutet 2D-Ansicht. Hier können Layout-Einstellungen für die drei verschiedenen 2D-Ansichten TOP, FRONT und LINKS vorgenommen werden.

Das Layout besteht aus verschiedenen Farbeinstellungen sowie Linienstärken für die angezeigten Geometrie-Objekte. Einerseits kann die **Hintergrundfarbe** der Ansichten eingestellt werden. Diese wird allerdings nur angezeigt, wenn das Grid deaktiviert ist. Andererseits kann die **Farbe** der Gitternetz-Struktur der Geometrie-Objekte ausgesucht werden. Unabhängig davon können die **Außenkanten** der angezeigten Geometrie-Objekte angepasst werden. Ebenso können die **Linienstärken** der Gitternetz-Struktur und der Außenkanten den individuellen Bedürfnissen zugeschnitten werden.

Selektion

Im Unterpunkt *Selektion* kann das Layout der selektierten Geometrie-Objekte den spezifischen Anforderungen angepasst werden. Zum Layout gehört hier die **Farbe** der Selektionslinien und den Außenkanten sowie der **Linienstärke** der Selektionslinien und deren Außenkanten. Auch hier sind die Außenkanten separat von den Selektionslinien einstellbar.

Licht

Bei *Licht* kann die **Maximale Anzahl von Lichtern** eingestellt werden. Dieser Wert gibt an, wie viele verschiedene Lichtquellen in der Welt existieren dürfen.

Bounding Volumes

In diesem Bereich lässt sich die Farbe der Bounding Volumes anpassen.

3D-Ansicht

Mit *3D-Ansicht* ist die Perspektiv-Ansicht gemeint. Ebenso wie bei der 2D-Ansicht kann hier das Layout dieser Ansicht angepasst werden.

Das Layout bezieht sich hier allerdings nur auf die **Hintergrundfarbe** der Perspektiv-Ansicht.

Objekte

Bei *Objekte* können der **Translationsmodus** und die **Detailstufe** des Objekts individuell eingestellt werden.

Der **Translationsmodus** gibt an, in welchem Koordinatensystem (global oder lokal) die Objekte verschoben werden. Hier kann in einem Drop-down Menü zwischen *Global*, *Lokal* und *Beides* gewählt werden.

Global bedeutet in diesem Zusammenhang, dass die Geometrie-Objekte in der Welt und deren Koordinatensystem verschoben wird. Lokal bedeutet... Die **Standard Gridoberfläche** gibt die Detailstufe der Objekte an, in wie vielen Schritten die Objekte unterteilt werden.

Animation

Bei *Animation* kann man Einstellungen, die die OH-Animation betreffen, vornehmen. Dazu gehören **Farbeinstellungen** und die **Größe** der JointPoints, sowie einen Wert, der die Zeit angibt, wann sich die **Animation aktualisiert**. Ein JointPoint ist ein Verbindungspunkt, der gibt an, wie ein Geometrie-Objekt relativ zum Elternobjekt positioniert und orientiert ist. Die Farbeinstellungen des JointPoints beziehen sich auch auf die selektierten JointPoints. Es ist sinnvoll, dass die JointPoints größer als normale Vertices dargestellt werden, um sie besser unterscheiden zu können.

Trigger

Unter dem Punkt *Trigger* können die **Farbe** und die **Linienstärke** des Triggerbereichs unabhängig voneinander angegeben werden. trigger sind bereiche, die beim Betreten oder verlassen der Spielerfigur bestimmte Ereignisse auslösen können.

Maßeinheit

Hier kann man die *Maßeinheit* einstellen. Dabei kann in einem Drop-down-Menü aus folgenden Einheiten gewählt werden: **Millimeter**, **Centimeter**, **Meter** (SI-Unit), **Kilometer**, **Inch**, **Foot**, **Yard**, **Statute Mile** und **Nautical Mile**. Diese Einheiten beziehen sich auf die Translationswerte und Größe der Geometrie-Objekte.

Editorlayout

Bei *Editorlayout* kann zwischen zwei verschiedenen **Iconsets** gewählt werden. Diese Icons sind in den verschiedenen Werkzeugleisten zu finden.

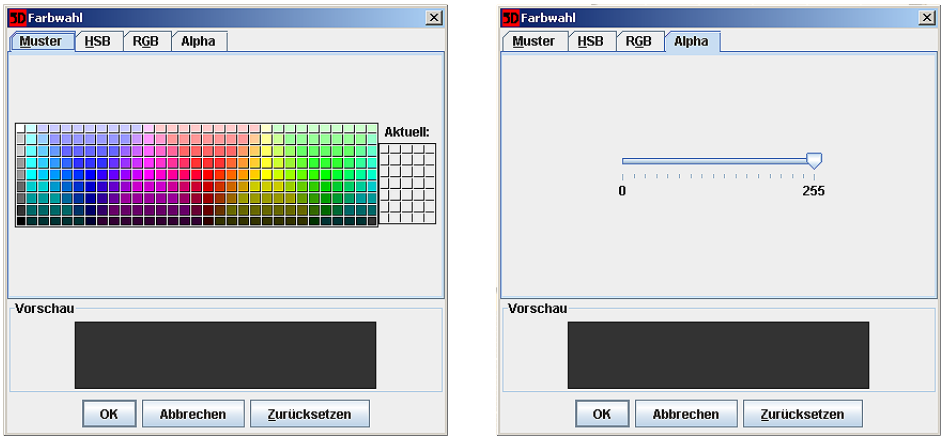
2.3.5 Eigene GUI-Komponenten

Da die Auswahl an standardmäßig vorhandener GUI-Komponenten im Paket `javax.swing` für einige Komponenten unseres Editors unzureichend ist, haben wir eigene GUI-Komponenten entwickelt. Diese angepassten Komponenten befinden sich im Paket `yage3d.editor.gui.utilities`.

Dazu gehören die Farbauswahlpanels (`ColorHexField`), die u.a. im Eigenschaftenfenster der Licht-Objekte zu finden sind: Dieses Panel enthält ein Textfeld, in dem der

Abbildung 2.13: `ColorHexField`

Hexadezimalcode der Farbe sowie der Alpha-Wert angezeigt ist. Die ersten sechs Zeichen entsprechen dem Farbcode, die letzten beiden Zeichen dem Alpha-Wert. Ein niedriger Alpha-Wert veranlasst, dass die Farbe transparent wird- Bei einem Wert von „00“ ist die Farbe komplett (zu 100%) transparent, bei „FF“ ist sie zu 0%, also gar nicht, transparent. Des weiteren befindet sich neben dem Textfeld ein Farbwahl-Button. Dieser öffnet ein angepasstes Farbauswahlfenster, den `ExtendedColorChooser`. Der Unterschied zu dem in Java verfügbaren `ColorChooser` besteht im wesentlichen aus dem veränderten Vorschau-Fenster und einem weiteren Reiter zum Alpha-Wert bestimmen:



Die in diesem Farbauswahl-Fenster eingestellten Farbwerte werden im Textfeld übernommen.

Im Eigenschaftenfenster der Geometrie-Objekte sind weitere angepasste GUI-Komponenten zu finden. Zum einen gibt es das `ExtendedJTextField`, zum anderen ein `LinkFiled`. Beides sind Textfelder mit unterschiedlicher Funktion:

Im `ExtendedJTextField` Wird der Objekt-Typ angezeigt. Dieses textfeld ist jedoch inaktiv, der der Typ eines geometrie-Objekts vom Benutzer so nicht editierbar ist.

Das `LinkField` zeigt die vorhandenen Oberflächen des Geometrie-Objekts an und hat eine besondere Eigenschaft. Wenn man auf dieses Textfled mit der linken Maustaste klickt, so wird automatisch die jeweilige Oberfläche selektiert und das entsprechende Eigenschaftenfenster öffnet sich, so dass die Oberfläche bearbeitet werden kann.

In dem eigenschaftenfenster für Oberflächen befindet sich die `ExtendedJTextArea`. Diese zeigt die Funktionen an, mit denen eine Oberfläche transformiert wurde. Auch dieser Bereich ist vom Benutzer nicht editierbar, um weitere Transformationen hinzuzufügen, muss er die Buttons neben dem Textbereich verwenden.

2.3.6 Hilfe

Die Hilfe wird global für den kompletten Editor angeboten und ist nicht vom Bearbeitungsmodus abhängig. Es gibt eine Kontext-sensitive Hilfe zur schnellen Problemlösung, die durch Drücken der Taste „F1“ aufgerufen wird. Ebenso gibt es eine traditionelle Hilfe, die im Editor im Hauptmenü aufgerufen werden kann. Die Hilfe-Seiten sind HTML-Dokumente, so dass diese Hilfe auch online einsehbar sein kann. Dem zugrunde liegt eine .css (Cascading Stylesheet)-Datei, um ein einheitliches Layout der Hilfe-Seiten zu garantieren.

Jede Hilfe-Seite hat ihr eigenes Thema, mit dem sie sich beschäftigt. Screenshots vereinfachen die einzelnen beschriebenen Arbeitsschritte, zudem gibt es auf jeder Seite der Hilfe Links zu verwandten Themen, falls die gesuchte Lösung nicht gefunden wurde.

Zur Verwaltung der Hilfe bietet das Paket `yage3d.editor.gui.help` jeweils eine Klasse zum Aufruf der (Kontext-sensitiven) Hilfe.

Die Hilfeseiten sind jeweils auf Deutsch und auf Englisch verfügbar. Die Anzeige wird durch die aktuell eingestellte Editor-Sprache manipuliert. Das bedeutet, wenn die aktuelle Editorsprache Englisch ist, werden die Hilfe-Seiten auch auf Englisch dargestellt. Zu dem gibt es eine Suchfunktion, mit der das Auffinden der benötigten Hilfe-Seite vereinfacht wird.

2.3.7 Normal-Modus

Der Normalmodus dient zur Gestaltung der verschiedenen Welten. Hier können die verschiedenen Objekt-Typen (Geometrie-, OH-, Licht-, Trigger- und Partikel-Objekte) erstellt und bearbeitet werden. Die Ansicht des Editors im Normal-Modus unterscheidet sich etwas von der des OH-Animationsmodus. Die einzelnen Fenster im `ObjectFrame` sind der jeweiligen Funtkionalität angepasst, ebenso gibt es in den verschiedenen Modi differenzierte Werkzeugleisten und Eigenschaftenfenster.

Werkzeugleisten

Es gibt Modus-spezifische Werzeugleisten, die hier nun für den Normal-Modus näher erläutert werden. Die Werkzeugleisten, die in beiden Modi die selbe Funktionalität bieten, wurden bereits im Abschnitt 2.3.2 beschrieben.

Variable Werkzeugleisten

Von den variablen Werkzeugleiste ist nur die **Objekt**-Werkzeugleiste in beiden Modi unterschiedlich. Diese Werkzeugleiste befindet sich, wie die anderen auch, unterhalb des Hauptmenüs.

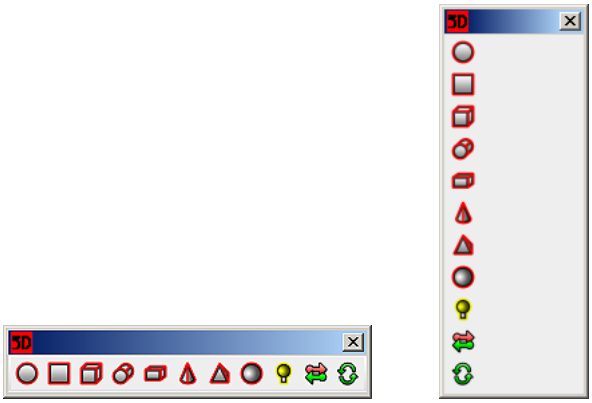
Wie bereits beschrieben, wird zum Ein- bzw. Ausblenden wird das Hauptmenü verwendet. Unter **Ansicht** → **Werkzeugleisten** kann die Objekt-Werkzeugleiste aktiviert bzw. deaktiviert werden (Abbildung 2.14).

Objekt-Werkzeugleiste

Die Objekt-Werkzeugleiste (Abbildung 2.14 dient zur Generierung der Geometrie- und Licht-Objekte.

Abbildung 2.14: `ObjectToolbar`

Man kann zum einen 2D-Objekte (Scheibe und Rechteck) erzeugen, zum anderen können hier auch 3D-Objekte per Knopfdruck erzeugt werden. Diese sind: der Würfel, der Zylinder, der Quader, der Kegel, die Pyramide und die Kugel. Diese Geometrie-Objekte können ebenfalls über das PopUp-Menü im Szenenbaum generiert werden. Zu dem kann nur über diese Werkzeugleiste ein Licht-Objekt in die Welt eingefügt werden. Der Button ganz rechts () dient zum Abdocken der Werkzeugleiste. Benutzt man diesen, wird die Werkzeugleiste horizontal in einem eigenständigen Fenster dargestellt:



Orientierungsmöglichkeiten der Werkzeugleiste

Dabei wird ein weiterer Button der Werkzeugleiste hinzugefügt ( ). Dieser verändert die Orientierung der Werkzeugleiste. Wenn sie zuvor horizontal dargestellt wurde, wird sie nun vertikal angeordnet. Andersherum funktioniert es analog. Um die Werkzeugleiste wieder an das Editor-Fenster zu binden schließt man einfach das Fenster mit der Werkzeugleiste. Anschließend befindet sich diese wieder unterhalb des Hauptmenüs.

Feste Werkzeugleisten

Die festen Werkzeugleisten sind, wie der Name schon vermuten lässt, fest in das Layout des Editors integriert. Im Normal-Modus befinden sich diese Werkzeugleisten im Szenenbaum- und im Browser-Fenster.

Szenenbaum Mit der Szenenbaum-Werkzeugleiste können Operationen ausgeführt werden, die den Szenebaum betreffen. Geometrie-Objekte können mit Hilfe dieser Werkzeugleiste gruppiert sowie ein- und ausgeblendet werden. Zudem können die Objekte im Szenenbaum umbenannt, gelöscht oder via CSG (Abschnitt 2.2.2) addiert oder subtrahiert werden.
Einzelne Objekte sowie Gruppen können versteckt werden, diese Objekte sind also nicht

Abbildung 2.15: `ScenetreeToolbar`

mehr sichtbar auch der Arbeitsfläche, was die Modellierung der Welt deutlich vereinfachen kann. Ebenso kann man Gruppen wieder sichtbar machen oder alle, bis auf das selektierte Objekt verstecken. Natürlich ist es auch möglich, sämtliche versteckten Objekte auf einmal sichtbar zu machen. Wie bereits erwähnt können Objekte auch via Buttondruck gelöscht oder umbenannt werden. Ebenso können die Objekte zu Gruppen zusammengefasst oder vorhandene Gruppierungen gelöscht werden. Die letzten beiden Buttons stehen für die CSG-Operationen „Add“ und „Substract“.

Browser

Im Browser werden die Ordner angezeigt, die sich in dem Projekt-Ordner befinden. Je nachdem, welcher dieser Ordner selektiert wird, wird am unteren Rand des Browsers eine passende Toolbar angezeigt.

Surface Transformation-Werkzeugleiste

Abbildung 2.16: `SurfaceTransformationToolbar`

Bei Selektion des Ordners **Surface Transformations** wird die Werkzeugleiste in Abbildung 2.16 angezeigt. Mit Hilfe dieser Toolbar können Displacementmaps aus .jpg-Dateien geladen oder zufällige Displacementmaps generiert werden. Weiterhin ist es möglich, eine Transformation mittels eines Strings (Textzeile) zu erzeugen. Natürlich sind vorhandene Transformationen auch mittels Button löschbar.

Weltbrowser-Werkzeugleiste

Ist der **Worlds**-Ordner angewählt, erscheint die Werkzeugleiste (Abbildung 2.17), mit der man neue Welten erstellen und vorhandene löschen kann.

Abbildung 2.17: `WorldbrowserToolbar`

Objektbrowser-Werkzeugleiste

Wenn der **Objects**-Ordner selektiert wurde, wird folgende Werkzeugleiste angezeigt.

Abbildung 2.18: `ObjectbrowserToolbar`

Die Buttons dienen zum Importieren weiterer Objekte sowie zum Löschen vorhandener Objekte.

Soundbrowser-Werkzeugleiste

Anhand der Soundbrowser-Werkzeugleiste (Abbildung 2.19), die bei Selektion des **Sound**-Ordners angezeigt wird, können zum einen Sound-Dateien verwaltet werden, zum anderen bietet sie Vorschau-Funktionen für diese Dateien.

Abbildung 2.19: `SoundbrowserToolbar`

Zur Verwaltung der Sound-Dateien können neue Ordner angelegt werden, einzelne Sound-Dateien oder komplette Ordner mit Sound-Dateien importiert werden. Zu dem können vorhandene Ordner und einzelne Sound-Dateien gelöscht werden.

Als Vorschaufunktion können Sound-Dateien abgespielt und beim Abspielen gestoppt werden. Weterhin kann man auch durch die Sound-Dateien navigieren, in dem die vorherige oder die nächste Sound-Datei gewählt werden können.

Texturbrowser-Werkzeugleiste

Bei Selektion des **Texture**-Ordners befindet sich am unteren Rand des Browser-Fensters die Texturbrowser-Werkzeugleiste (Abbildung 2.20). Wie auch beim Sound können hier

Abbildung 2.20: `TexturbrowserToolbar`

neue Unterordner erstellt und einzelzene Texturen sowie komplette Ordner importiert werden. Ebenfalls kann man komplette Unterordner und einzelne Textur-Dateien löschen.

Skriptbrowser-Werkzeugleiste

Ist der Scripts-Ordner ausgewählt, erscheint die Scriptbrowser-Werkzeugleiste im Browser.

Mit Hilfe dieser Werkzeugleiste können neue Jython-Skripte erzeugt und bereits

Abbildung 2.21: `SkriptbrowserToolbar`

vorhandene gelöscht werden.

Objekt-Fenster

Der `ObjectFrame` im Normal-Modus wird nochmals unterteilt. Im oberen Bereich befinden sich der Szenenbaum und der Projekt-Browser als „Tabbed Window“, das bedeutet, dass die unterschiedlichen Fenster übereinander liegen und über ein „Tab“ bzw. „Reiter“ anwählbar sind. Im unteren Bereich befinden sich die Eigenschaftenfenster, die in den Abschnitten 2.3.7 näher beschrieben werden.

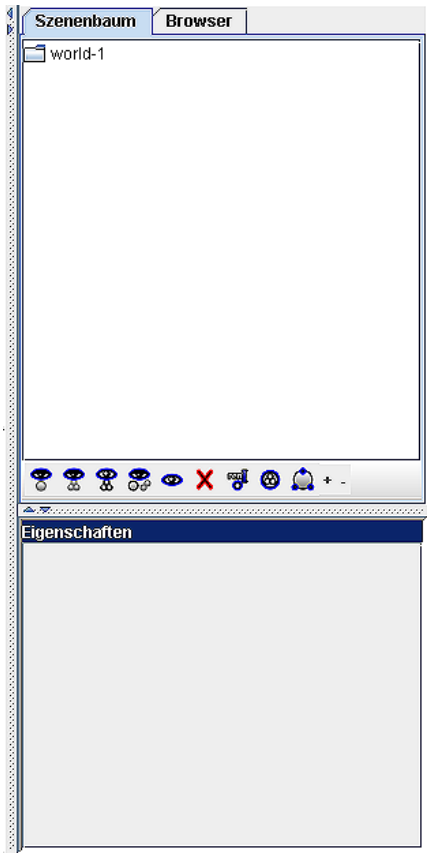


Abbildung 2.22: `ObjectFrame`

Der **Szenenbaum** (`SceneTree`), welcher von `javax.swing.JTree` abgeleitet wurde, enthält einen Baum, der die Welt repräsentiert. In diesem Baum befinden sich die Gruppen-, Objekt- und Lichtknoten. Diese Knoten können zum einen durch die Werkzeugleiste im unteren Browserbereich bearbeitet werden, zum anderen öffnet ein Rechts-Klick mit der Maus auf einem Knoten ein PopUp-Menü mit ausführbaren Anweisungen:

- **Neu** (Gruppenknoten, Terrain [erzeuge Karte, erzeuge Zufallskarte], OH-Objekt, Quader, Würfel, Kugel, Kegel, Zylinder, Pyramide, FreeHand Object, MD2 importieren, Rechteck, Scheibe, Trigger)
- Umbenennen

- in CSG-Objekt umwandeln
- Löschen
- OH-Object bearbeiten (nur beim OH-Objekt aktiv)

Man erkennt sehr leicht, dass die Menü-Einträge im Unterpunkt „Neu“ in verschiedene Bereiche aufgeteilt sind. Diese Bereiche sind neue Knoten, die nach Gruppen-, Objekt- und Trigger-Knoten geordnet sind. Die Objekt-Knoten sind ebenfalls strukturiert: Der erste Teil bezieht sich auf neue komplexe 3D-Objekte, anschließend werden die primitiven 3D-Objekte aufgelistet. Danach stehen Freihand-Objekte und Importe auf der Liste, gefolgt von den 2D-Objekten und zu guter letzt dem Trigger.

Durch „Umbenennen“ kann dem Knoten oder dem Objekt ein neuer Name zugewiesen werden. Die Namen sind pro Gruppenknoten eindeutig, sollte ein bereits existierender Name vergeben werden, so wird automatisch ein eindeutiger Name generiert. Dieses geschieht durch anhängen einer Tilde („∼“) und einer Zahl, welche die Häufigkeit des Namens angibt. Soll beispielsweise einem Quader der Name „Box“ zugewiesen werden, allerdings existiert schon ein Objekt in dem selben Gruppenknoten mit diesem Namen, so wird aus dem „Box“ ein „Box∼2“.

Der **Browser** erlaubt den Zugriff auf den Projekt-Ordner mit all seinen Unterordnern und den darin enthaltenen Dateien. Der Browser setzt sich aus einem Ordner-Baum und einem Vorschau- bzw. Ordner-Eigenschaftenfenster zusammen. Im linken Teil wird der Ordner-Baum angezeigt. Folgende Ordner sind im Projekt-Ordner immer vorhanden, es können aber auch eigene Ordner hinzugefügt werden: „Projekt Daten“ mit dem Unterordner „Surface Transformation“, „worlds“, „objects“, „sounds“, „textures“ und „scripts“.

Bei Anwahl des **Projekt**-Ordners, sowie des Ordners **Projekt-Daten**, erscheinen Projekteigenschaften im rechten Bereich des Browsers. Hier werden der Projektname, das Erstellungsdatum, sowie die Anzahl der Welten und den darin enthaltenen Objekten angezeigt.

Eigenschaftenfenster

Facade- Entwurfsmuster

Das Facade-Entwurfsmuster (Facade = Fassade) wird benutzt, wenn einfache Schnittstellen zu einem komplexen Subsystem angewendet werden sollen, vor allem, wenn es zahlreiche Abhängigkeiten zwischen einem Klienten und dem Subsystem gibt [4]. Die Aufgabe des Facade-Entwurfsmusters ist demnach die Kapselung systemnaher Klassen, denen eine objektorientierte Schnittstelle zur Kommunikation angeboten wird [1].

Durch die Benutzung des Facade-Entwurfmusters werden die Abhängigkeiten zwischen dem Klienten (Auftraggeber, der einen bestimmten Dienst in Anspruch nehmen möchte) und dem Subsystem deutlich vermindert [2]. Die Zugriffe auf ein Subsystem werden durch ein Interface, die sogenannte Fassade, gebündelt [3]. Durch diese Bündelung werden die einzelnen Komponenten des Subsystems vom Klienten entkoppelt, wodurch Portabilität und Unabhängigkeit des Subsystems gefördert werden [4].

Das Subsystem wird in Teilkomponenten organisiert, wobei die Fassade den Zugriff für jede Teilkomponente definiert. Somit wird die Struktur des Subsystems vereinfacht, da die Kommunikation ausschließlich über die Fassade geschieht.

Die Fassade „weiß“, welche Klassen des Subsystems für welche Aufgaben zuständig sind. Sie leitet die Nachrichten vom Klienten an das zuständige Paket weiter ohne neue Funktionalität zu definieren.

Die Klassen innerhalb des Subsystems führen die von der Fassade zugewiesenen Aufgaben durch, wissen allerdings nicht, dass solch eine Fassade existiert.[4]

Die Vorteile dieses Entwurfmusters liegen auf der Hand. Der Code wird kompakter durch höhere Abstraktion und objektorientierte Schnittstellen, was auch zur besseren Verständlichkeit des Codes beiträgt. Die Modularität wird durch die Gruppierung der Funktionen in den Klassen des Subsystems und die Organisationstruktur der Klassen deutlich verbessert. Ebenso wird die Portablitität verstärkt durch diese Organisation und die Implementierungsverfahren [1].

Realisierung der Fassade

Unsere Eigenschaftenfenster in der GUI sind nach dem Facade-Entwurfsmuster aufgebaut. Sämtliche Eigenschaftenfenster für die verschiedenen Objekttypen liegen in dem separaten Paket `yage3d.editor.gui.propertiespanel`. Die Schnittstelle zu diesen Eigenschaftenfenster wird durch die Klasse `PropertiesPanelFacade` realisiert. Dieses Interface stellt drei Methoden zum Zugriff auf die Eigenschaftenfenster zur Verfügung, die später noch näher erläutert werden.

Intern gibt es von jedem Eigenschaftenfenster-Typ eine Instanz in der `PropertiesPanel-Panel`.

Alle Fenster sind vom Typ `AbstractPropertiesPanel` und können daher Methoden zur Strukturierung der einzelnen Buttons und Labels benutzen, um ein einheitliches Layout zu garantieren. Es können verschiedene Gruppen von GUI-Elementen definiert werden. Die Gruppen vereinfachen den Umgang mit Swing-Komponenten, indem sie diese in bis zu drei dynamischen Spalten anordnen. Zu jedem Eigenschaftenfenster gehört auch ein angepasster Listener, der im Controller-Teil unserer Software zu finden ist.

Die Struktur und die Polymorphie unseres Subsystems wird durch Abbildung 2.23 noch deutlicher.

Der gesamte Zugriff auf das Eigenschaftenfenster geschieht durch 3 Funktionen der Klasse `PropertiesPanelFacade`. Diese Funktionen sind `showProperties(Object)`, `update()` und `updateTransformation()`.

showProperties(Object)

Diese Funktion zeigt das Eigenschaftenfenster eines bestimmten Objekt-Typs an. Dieser

Abbildung 2.23: Polymorphie der Eigenschaftenfenster

Funktion können alle Variationen an Parametern übergeben werden, es macht keinen Unterschied, ob sie einen Gruppenknoten (GroupNode) oder einen anderen Knoten übergeben bekommt; Ebenfalls können komplexe Objekte wie beispielsweise DisplacementmapSurfaceTransformations übergeben werden. Die Methode erkennt automatisch den Typ des übergebenen Objekts und zeigt das entsprechende Eigenschaftenfenster an.

update()

Diese Funktion aktualisiert die Eigenschaftswerte des aktuell angezeigten Objekts. Wenn der Name eines Knotens geändert wurde, wird - nach Aufruf dieser Methode - der neue Name im Eigenschaftenfenster angezeigt. Ebenso verhält es sich bei Änderungen der anderen Werte. Werden nur die Positionswerte des Objekts sowie dessen Orientierung, also die Rotationswerte, verändert, kann statt dessen die `updateTransformtion()` benutzt werden.

updateTransformation()

Diese Funktion verhält sich ähnlich wie `update()`. Sie wird aufgerufen, wenn sich die Position des Objekts oder dessen Orientierung im Raum geändert hat. Diese Methode ist allerdings wesentlich schneller als die `update()`-Methode, und da der `CanvasListener` bei Mausbewegungen sehr oft Objektverschiebungen durchführen kann, macht diese Methode hier mehr Sinn.

`PropertiesPanelFacade` ist nur für das Anzeigen der Eigenschaftenfenster zuständig. Welches Fenster angezeigt wird, entscheiden u.a. der SelectionController und der ProjektBrowser, welche direkt auf die Fassade zugreifen.

`AbstractPropertiesPanel` definiert die Struktur des Eigenschaftenfensters und ermöglicht das Einfügen von gruppierten GUI-Elementen in den verschiedenen Eigenschaftenfenstern. Diese Klasse enthält außerdem eine Funktion zum Hinzufügen der Event-Listener.

`AbstractNodePropertiesPanel` ist die Oberklasse aller Eigenschaftenfenster für die

verschiedenen Knotentypen. Diese Knotentypen sind:

- Gruppenknoten (GroupNode)
- Objektknoten (ObjectNode)
- Lichtknoten (LightNode)
- Triggerknoten (TriggerNode)

Sie enthält Eigenschaftsregler für die Position der Objekte in den Knoten und für die Rotation, das heißt die Orientierung der Objekte im Raum. Ebenso beinhaltet die Klasse Textfelder für den Namen und den Objekt-Typ.

Je nach Objekt in dem Knoten sind auch andere Einstellungen denkbar, beispielsweise die Größe und die Detailstufe, die ebenfalls über Eigenschaftsregler zugewiesen werden können.

Dem Lichtobjekt können weitere Licht-spezifische Eigenschaften übergeben werden, wie Intensität oder die Art und die verschiedenen Farbwerte des Lichts.

`AbstraktSurfaceTransformationPanel` ist die Oberklasse der Eigenschaftenfenster für Surface-Transformationen. Surfaces können über mathematische Formeln oder Heightmaps transformiert werden.

`AbstractDisplacementMapSurfaceTransformationPanel` ist abgeleitet von der bereits beschriebenen Klasse `AbstraktSurfaceTransformationPanel`. Sie ist die Oberklasse für die Eigenschaftenfenster der DisplacementMaps und beinhaltet unter anderem Regler für die Höhe und die Breite der Heightmap. DisplacementMaps können zufällig oder aus Bild-Dateien generiert werden.

2.3.8 OH-Modus

Der OH-Animationsmodus dient zur Objekthierarchischen Animation von bereits in einer Welt erstellten Objekten. In diesen Modus kann man durch „OHObjekt bearbeiten“ aus dem PopUp-Menü, aufgerufen durch linken Maustastenklick, des OHGeoObjects wechseln.

Die Arbeitsfläche und die Hauptmenüleiste stimmen mit dem Normal-Modus überein, unterschiedlich sind hier einige Werkzeugleisten und auch der `ObjectFrame`.

Werkzeugleisten

Die einzige **variable** Werkzeugleiste, die sich vom Normal-Modus unterscheidet, ist die `ObjectToolbar`:

Abbildung 2.24: `ObjectToolbar` im OH-Modus

Im OH-Modus ist es demnach nicht mehr möglich, neue Objekte einzufügen. Wenn neue Objekte in das OH-Objekt importiert werden sollen, so muss man den ObjektBrowser benutzen. Soll ein Objekt, das noch nicht in der Welt existiert, importiert werden, so muss man in den Normal-Modus wechseln und es dort einfügen.

Bei den festen Werkzeugleisten fällt die `AnimationToolbar` unterhalb der Arbeitsfläche auf:

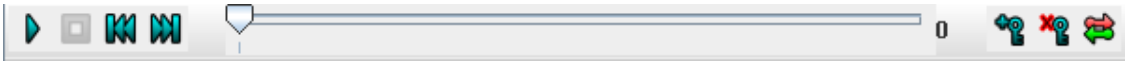


Abbildung 2.25: `AnimationToolbar`

Mit dieser Werkzeugleiste kann die aktuelle Animation abgespielt und eine laufende Animation gestoppt werden. Ebenso kann hier zum Anfang bzw. zum Ende der Animation gesprungen werden. In der Leiste wird der Vorsprung der Animation angezeigt sowie die Anzahl Keyframes. Jeder Keyframe kann über die Leiste ausgewählt werden. Zudem können Keyframes hinzugefügt bzw. entfernt werden.

Eine weitere Werkzeugleiste befindet sich im Animationsbrowser. Mit ihr können neue Animationen hinzugefügt oder bestehende Animationen gelöscht werden:

Abbildung 2.26: `AnimationBrowserToolbar`

Objektfenster

Der `ObjectFrame` befindet sich auch im OH-Modus am rechten Rand des Editorfensters. Auch hier wird er nochmals in zwei Bereiche unterteilt. Im oberen Bereich befinden sich die Objekthierarchie, der Objektbrowser und der Animationbrowser als „Tabbed Window“. Im unteren Bereich befindet sich ein Eigenschaftenfenster. Diese unterschiedlichen Bereiche sollen im folgenden näher erläutert werden.

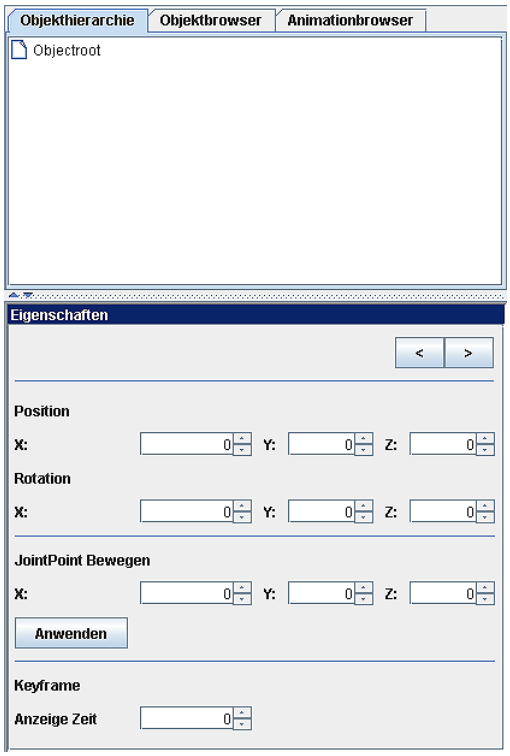


Abbildung 2.27: `ObjectFrame` im OH-Modus

Der **Objekthierarchiebaum** (`OHSceneTree`), welcher von `javax.swing.JTree` abgeleitet wurde, enthält einen Baum, der die Objekthierarchie innerhalb des OH-Objekts repräsentiert. In diesem Baum befinden sich die Gruppen- und Objekt-Knoten. Diese Knoten können im Objektbrowser (`ObjectBrowser`) hinzugefügt werden.

Der **Objektbrowser** enthält eine Liste mit den Objekten, die sich in der Welt befinden. Diese Objekte wurden zuvor im Normal-Modus erzeugt und können entweder per Doppelklick in die Objekthierarchie eingefügt werden oder mittels eines PopUp-Menüs, aufgerufen durch die rechte Maustaste. Das PopUp-Menü beinhaltet zwei Einträge: „Objekt importieren“ und „Objektreferenz importieren“. Unter *Objekt importieren* wird verstanden, dass dieses Objekt unabhängig von weiteren Objekten dieses Typs in das OH-Objekt eingefügt wird. Werden auf diese Weise zwei Quader in das OH-Objekt importiert und bei einem ändert sich die Größe, hat diese Veränderung keinerlei Auswirkungen auf den zweiten Quader. Anders bei der *Objektreferenz*: Werden zwei Quader als Referenzen in das OH-Objekt importiert und einer von ihnen wird verändert, so wirken sich diese Veränderungen auch auf den zweiten Quader aus.

Der **Animationsbrowser** zeigt eine Liste mit den vorhandenen Animationen an. Hier können über eine Werkzeugleiste am unteren Rand neue Animationen hinzugefügt bzw. vorhandene Animationen gelöscht werden. Wird eine neue Animation hizugefügt, öffnet sich der `AnimationNameDialog`, in dem der Name für die neue Animation festgelegt werden muss.

Dieser Name anschließend dann in den `AnimationSettings` gespeichert und im Animationbrowser angezeigt.

Eigenschaftenfenster

Das Eigenschaftenfenster unterscheidet sich von dem im Normal-Modus, da hier die Verwendung des Facade-Entwurfmusters nicht notwenig ist. Im OH-Modus gibt es nur dieses eine Eigenschaftenfenster, in dem alle notwengigen Einstellungen vorgenommen werden können.

Ist ein **Geometrie-Objekt** selektiert, kann es im Eigenschaftenfenster relativ zu seinem Eltern-Objekt verschoben und rotiert werden. Dafür stehen Eigenschaftenregler sowie

Textfelder für die einzelnen Koordinaten (für die Verschiebung, auch Translation genannt) sowie für die einzelnen Achsen (für die Rotation) zur Verfügung.

Der JointPoint des Geometrie-Objekts kann über weitere Eigenschaftenregler innerhalb dieses Geometire-Objekts verschoben werden. Dabei verschiebt sich intern nicht der JointPoint, sondern die Eckpunkte des Geometrie-Objekts werden verschoben, da der JointPoint immer am Ursprung des Geometrie-Objekts liegt. Eine Translation des JointPoints ist also eine Verschiebung des Geometrie-Objekt-Urpsrungs.

In dem Eigenschaftenfenster kann auch die Zeit (Display Time) festgelegt werden, die der jeweilige **Keyframe** angezeigt werden soll (wie viele Millisekunden also zwischen diesem und dem nächsten Keyframe vergehen). Diese Zeitspanne wird zur Interpolation zwischen den Keyframes verwendet.

2.4 Controller

Die Controller-Schicht wird in unserer Software durch diverse Listener und Controller realisiert. Die Listener sind an GUI-Komponenten gekoppelt, wobei die Controller die verschiedenen Modi des Editors überwachen und kontrollieren. Der Controller bietet demnach die Schnittebene zwischen dem Model und der View.

Es gibt zum einen den `MainController`, der den Normal-Modus des Editors überwacht. Der `OHContoller` übernimmt das Monitoring des OH-Animationsmodus. In beiden Modi läuft zu dem noch der `SelektionController`, der den jeweiligen Selektionsmodus (Objekt, Fläche oder Vertex) kontrolliert.

Nahezu jede GUI-Komponente benötigt einen eigenen *Listener*. So hat zum Beispiel jede Werkzeugleiste ihren eigenen Beobachter, der bei Interaktion reagiert und die gewünschten Aktionen durchführt. Diese Listener sind in diesem Paket zu finden.

Desweiteren kümmert sich der Controller auch um die Vergabe von Programm-internen Identifikationen, für die der `IDGenerator` zuständig ist. Ebenso stellt der Controller einen `ExtendedXMLAdapter` zur Überwachung der Speicherung zur Verfügung.

2.5 PlugIns

2.5.1 PolygonEditor

Der `PolygonEditor` ist gewissermaßen ein PlugIn und wird in einem separaten Fenster gestartet (Abbildung 2.28). Mit Hilfe dieses Editors lassen sich Polygone (`PolygonObject` zeichnen und extrudieren (auseinanderziehen). Dieser besteht aus einer Werkzeugleiste (`PolygonToolBar`)

und einer Arbeitsfläche.

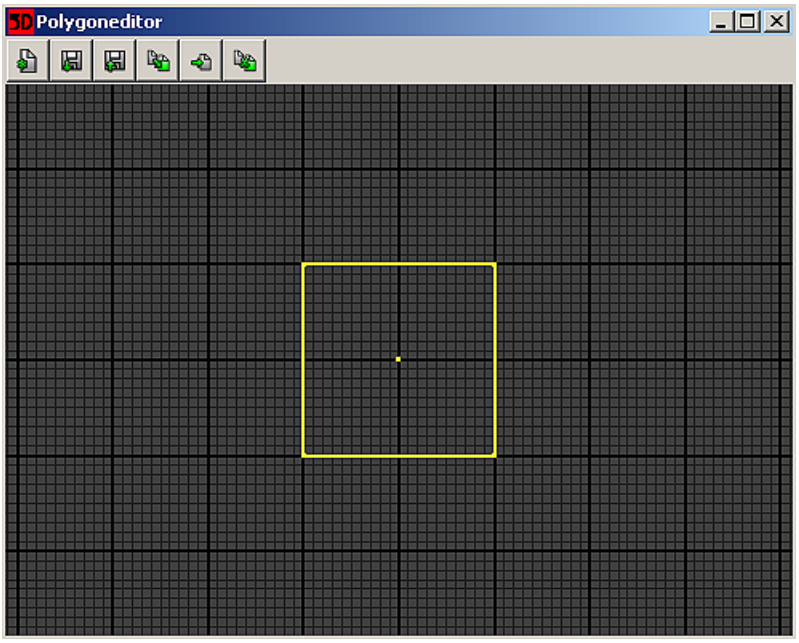


Abbildung 2.28: PolygonEditor

Die Arbeitsfläche beinhaltet standradmäßig ein rechteckiges Polygon und einen Punkt, dem sogenannten Relationspunkt. Die Eckpunkte des Polygons (`PolygonEdge`) können verschoben werden, so dass individuelle, viereckige Grunsformen entstehen.

Durch die Werkzeugleiste lassen sich neue Polygone erzeugen, vorhandene öffnen zum Bearbeiten und fertiggestellte Polygone speichern. Zudem werden drei verschiedene Extrusions- bzw. Rotations-Operationen durch die Icons zur Verfügung gestellt.

Extrusion zur Kante (Extrude to bevel)

Die Extrusion zu einer Kante bedeutet, dass alle Punkte des Polygons auf den Relationspunkt zulaufen, diesen aber nie erreichen. Das Polygon wird also in die Tiefe gezogen und dabei verkleinert. Dabei muss der Relationspunkt außerhalb des Polygons liegen. Es öffnet sich der `PolygonExtrudeToBevelDialog`, in dem verschiedene Einstellungen vorgenommen werden können. Dazu gehören der Name des Polygons sowie zwei Tiefenangaben für die Extrusion.

Der **Name** des Polygons wird übernommen und im Szenenbaum angezeigt.

Tiefe gibt die Z-Koordinate des Punktes an, auf den das Polygon zuläuft.

Kantentiefe gibt die Z-Koordinate des Polygons an, es ist also der Wert, bei dem die Extrusion endet.

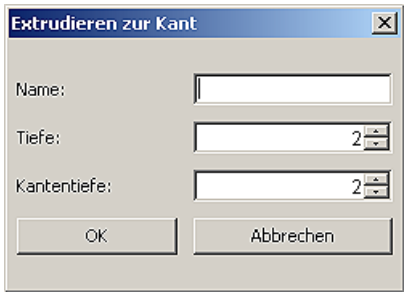


Abbildung 2.29: `PolygonExtrudeToBevelDialog`

Mit dem OK-Button wird das Objekt in die Welt eingefügt. Der Polygoneditor bleibt dennoch geöffnet, falls weitere Polygonobjekte erzeugt werden sollen. Der Abbrechen-Button verwirft das erstellte Polygonobjekt, so dass weiterhin im Polygoneditor gearbeitet werden kann.

Extrusion zu einem Punkt

Die Extrusion zu einem Punkt bedeutet, dass alle Eckpunkte des Polygons auf den Relationspunkt zulaufen. Dabei muss sich der Relationspunkt außerhalb des Polygons befinden. Es öffnet sich der `PolygonExtrudeDialog`, in dem der Name des Polygons und die Tiefe angegeben werden können.

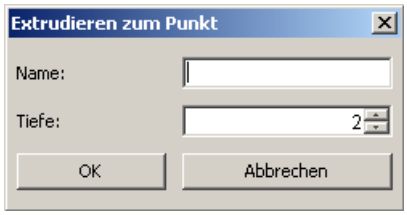


Abbildung 2.30: `PolygonExtrudeDialog`

Auch hier wird der **Name** auch im Szenenbaum übernommen.

Tiefe gibt die Z-Koordinate des Polygons an, also wie weit es in die Tiefe gezogen wird.

Mit dem OK-Button wird das Objekt in die Welt eingefuegt. Der Polygoneditor bleibt dennoch geöffnet, falls weitere Polygonobjekte erzeugt werden sollen. Der Abbrechen-Button verwirft das erstellte Polygonobjekt, so dass weiterhin im Polygoneditor gearbeitet werden kann.

Rotation um einen Punkt

Die Rotation um den Punkt bedeutet, dass das Polygon um eine parallele Gerade zur Y-Achse rotiert wird. Diese Gerade wird durch den Relationspunkt repräsentiert. Der Relationspunkt muss hierbei links oder rechts außerhalb des Polygons liegen. Es öffnet sich der , in dem der Name des Polygons angegeben sowie Rotationseinstellungen vorgenommen werden können.

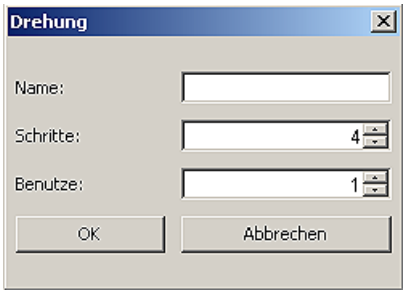


Abbildung 2.31: `PolygonRevolveDialog`

Der **Name** des Polygon-Objekts gibt gleichzeitig den Namen dieses Objekts im Szenenbaum an.

Schritte gibt an, in wie vielen Schritten die 360° einer kompletten Rotation unterteilt werden. Bei einer Schrittzahl von 36 entspricht jeder Schritt einem Winkel von 10°, bei 5 Schritten entspricht jeder Schritt einem Winkel von 72°.

Benutze gibt an, wie viele der angegebenen Schritte benutzt werden sollen, um das Objekt zu generieren. So können auch halb-runde Polygonobjekte erstellt werden. Wird beispielsweise bei Schritte 36 angegeben, und davon sollen nur 3 benutzt werden, so erhält man ein Polygonobjekt, das um 30° rotiert wurde.

Mit dem OK-Button wird das Objekt in die Welt eingefuegt. Der Polygoneditor bleibt dennoch geöffnet, falls weitere Polygonobjekte erzeugt werden sollen. Der Abbrechen-Button verwirft das erstellte Polygonobjekt, so dass weiterhin im Polygoneditor gearbeitet werden kann.

2.5.2 Script-Editor

Der `Scripteditor` ist ebenfalls ein PlugIn und wird in einem separaten Fenster gestartet (Abbildung 2.32). Mit Hilfe dieses Editors können Jython-Skripte erstellt werden. Dieser Editor besteht aus einer Werkzeugleiste, (`ScriptediotrToolbar`), einem Menü (`ScripteditorMenubar`)

und einer Arbeitsfläche.

Die Arbeitsfläche besteht aus einem editierbaren Textfeld, welches standardmäßig leer geöffnet wird. In diesem Textfeld können die Jython-Skripte angegeben werden.

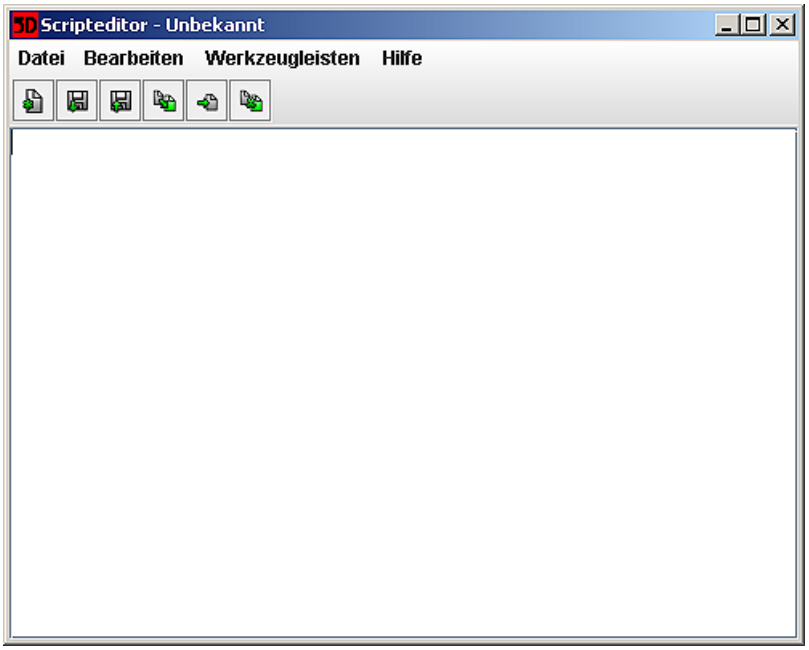


Abbildung 2.32: SkriptEditor

Durch die Werkzeugleiste lassen sich neue Jython-Skripte erzeugen, vorhandene zum Bearbeiten öffnen und fertiggestellte Skripte speichern. Zudem werden drei verschiedene Operationen durch die Icons zur Verfügung gestellt.

Das Hauptmenü des Editors beinhaltet die Einträge **Datei**, **Bearbeiten**, **Werkzeugleisten** und **Hilfe**.

Datei

Unter diesem Punkt befinden sich Einträge zur Dateiverwaltung.

Berabeiten

Hier befinden sich Bearbeitungsoperationen.

Werkzeugleisten

bei Werkzeugleisten kann die werkzeugleiste aktiviert bzw. deaktiviert werden.

Hilfe

Hier kann die Hilfe aufgerufen werden. *To be continued*

3 Engine

In diesem Kapitel wird der Aufbau der Engine näher erläutert.

3.1 Kollision

In dem Paket `yage3d.engine.collision` befinden sich die Klassen, die für Kollisionskontrolle nötig sind. Hier befinden sich Klassen zur Raumaufteilung und unterschiedliche Bounding Volumes.

3.1.1 Bounding Volumes

Bounding Volumes sind Hüllkörper, in Yage entweder Quader oder Kugeln, die ein Objekt umschließen. Diese Hüllkörper werden zur Kollisionsberechnung verwendet. Egal wie komplex das Objekt ist, das einfache Bounding Volume umschließt es komplett. Beim Kollisionstest werden demnach nur einfache Objekte, eben Quader oder Kugeln, auf Kollision überprüft. Das ist weitaus performanter als ein Polygon-genauer Test der einzelnen Objekte.

Es gibt verschiedene Arten von Bounding Volumes, die jeweils ihre Vorteile, aber auch Nachteile mit sich bringen.

Folgende Bounding Volume-Typen werden in Yage verwendet:

- `BVNULL: Kein Bounding Volume, Kollisionstests verlaufen immer negativ.`
- `BVBS: Bounding Sphere: Kugelförmiges Bounding Volume`
- `BVOBB: Oriented Bounding-Box: Quaderförmiges Bounding Volume`

BVBS

Bei der Bounding Volume Bounding Sphere (BVBS, kurz BS) handelt es sich um ein kugelförmiges Bounding Volume.

Vorteile:

Die Kollisionstest zwischen Bounding Spheres sind sehr schnell. Diese Art Bounding Volume eignet sich für großflächige Kollisionstests, oder für Abfragen, welche Objekte im Frustum der Kamera liegen. Ebenfalls von Vorteil ist, dass dieses Bounding Volume nicht neu berechnet werden muss, wenn das umschlossene Objekt seine Orientierung ändert.

Nachteile:

Für die meisten Objekte stellt eine Kugel nur eine schlechte Approximation der Objektform dar.

BVAABB Bei der Bounding Volume Axis Aligned Bounding Box (BVAABB, kurz AABB) handelt es sich um einen Quader, dessen Achsen parallel zu den Weltkoordinatenachsen verlaufen. AABBs werden nur intern zur groben Unterteilung des Raumaufteilungsbaumes verwendet, also zur Bestimmung dessen, welche Objekte soweit in der Nähe liegen, dass sich ein Kollisionstest lohnen könnte. AABB-Kollisionstest sind zwar sehr schnell, aber für eine Approximation eines Kollisionstest zweier Objekte sind AABBs i. d. R. viel zu ungenau (schlechte Fülleffizienz) und unflexibel (müssen nach Rotationen neu berechnet werden).

BVOBB

Bei der Bounding Volume Oriented Bounding Box (BVOBB, kurz OBB) handelt es sich um ein Quaderförmiges Bounding Volume, dessen Ausrichtung im Raum frei ist, das also beliebig rotiert werden kann.

Vorteile: Bei sehr vielen Objekten (insbesondere bei Autos oder Wänden) erweist sich eine OBB als sehr gute Annäherung an die eigentliche Objektgeometrie. Da die OBB frei orientierbar ist, kann sie zusammen mit dem Objekt gedreht werden, ohne neu berechnet werden zu müssen.

Nachteile:

Der Kollisionstest zwischen OBBs ist vergleichsweise rechenintensiv: Im Worst Case müssen bis zu 15 Projektionsachsen auf Intervallüberlappung getestet werden.

Eine komplexe Geometrie-Objekt-Gruppe, welche aus mehreren Geometrieobjekten zusammengestellt ist, kann für jedes Geometrieobjekt ein eigenes Bounding Volume besitzen. Ein Tisch beispielsweise, der aus fünf Quadern besteht (vier Tischbeine und eine Tischplatte), kann für jedes dieser Quader eine eigenen OBB besitzen.

3.2 objects

Die Objekte der Engine sind vereinfachte bzw. veränderte Varianten ihrer Editor-Pendants, die auf diverse Metainformationen verzichten können. Zudem gibt es weitere Engine-Objekte, die so nicht im Editor-Teil dieser Anwendung vorkommen.

Nachfolgend werden zunächst die Engine-Objekte erläutert, die auch im Editor zu finden sind. Anschließend werden die Objekte beschrieben, die nur in der Engine zu finden sind.

3.2.1 Engine-Objekte, zu denen es erweiterte Editor-Objekte gibt

geoobjects

In dem Paket `yage3d.engine.objects.geoobjects` sind neben den Geometrie-Objekten auch die Objekte, die für Animationen benötigt werden, zu finden.

Geometrie-Objekte

Als Geometrie-Objekte benötigt unsere Engine nur ein `BaseGeoObject`, von dem das `StaticGeoObject` sowie das `KeyframeAnimatedGeoObject` und auch das `OHGeoObject` abgeleitet werden.

Die Klasse `BaseGeoObject` legt fest, dass die abgeleiteten Klassen ein BoundingVolume sowie Attribute zum Zeichnen besitzen. Diese Attribute sind, ob das Geometrie-Objekt gezeichnet werden soll und ob es transparent ist. Zudem besitzt jedes Geometire-Objekt einen String, der den Dateinamen enthält, unter dem dieses Objekt gespeichert wird.

Das `StaticGeoObject` repräsentiert alle statischen Geometrie-Objekt-Typen des Editors. Dieser Typ wird durch eine *Displayliste* im statischen Geo-Objekt festgelegt. In einer Displayliste befinden sich alle Informationen über die Geometrie des Objekts, so dass das Objekt viel schneller gezeichnet werden kann. Jedes statische Geometrie-Objekt ist, wie auch im Editor, in der Lage, sich selbst zu zeichnen.

Das `KeyframeAnimatedGeoObjekt` repräsentiert alle Objekte, die durch eine Keyframe-Animation animiert sind. Jedes Keyframe-animierte Geometrie-Objekt kann sich selbst zeichnen, weiß, ob es animiert ist oder nicht und kennt seinen Animator (siehe Absatz Animation). Desweiteren sind alle für die Animation notwendigen Keyframes in diesem Objekt gespeichert. Für die Keyframe-Animation dieses Objekts steht auch das von OpenGL zum Zeichnen benötigte `GLDrawable`-Objekt in dem Keyframe-animierten Geometrie-Objekt zur Verfügung.

Das `OHGeoObject` ist etwas anders aufgebaut. In ein OHGeoObject können Geometrie-Objekte importiert werden, die in diesem OHGeoObject objekt-hierarchisch animiert werden. In diesem Zusammenhang sprechen wir von der OH-Animation (siehe Absatz OH-Animation). Das OHGeoObject ist in der Lage, alle in ihm entwahltenen BaseGeoObjects zu zeichnen. Dabei wird der Hierarchiebaum der Geometrie-Objekte rekursiv durchlaufen, so dass das OHGeoObjekt nur den Wurzelknoten des Baumes kennen muss. Auch die OH-Animation wird durch Keyframes gesteuert. Daher kennt das OHGeoObjekt auch seine Keyframes und den Animator, um die OH-Animation laufen zu lassen.

Die Geometrie-Objekte bestehen aus einem Netz, das aus Dreiecken besteht. Diese Dreiecke heißen Faces, die durch die Klasse `Face` realisiert werden. In den Faces sind die IDs der

zum Face gehörenden Eckpunkte (Vertices) sowie die IDs der zu den Vertices gehörenden Normalen gespeichert. Die Vertices werden durch die Klasse `Vertex` realisiert, wobei ein Vertex seine Koordinaten enthält. Die Normalen werden durch die Klasse `Normal` repräsentiert. Eine Normale besteht aus einem Vektor, der die jeweiligen Richtungsangaben für die drei Achsen enthält.

Im Paket `yage3d.engine.objects.geoobjects` sind auch alle notwendigen Objekte für die Keyframe- und die OH-Animation zu finden.

Keyframe-Animation

Bei der Keyframe Animation werden für die gewünschte Animation die relevanten Bewegungssequenzen in einzelnen Keyframes modeliert. Um aus diesen einzelnen Keyframes eine flüssige Bewegung zu erzeugen, wird zwischen gleichen Vertices verschiedener Keyframes der aufeinander folgenden Schlüsselposition interpoliert. Interpolieren bedeutet, dass die Position in Abhängigkeit von der Zeit, zwischen dem letzen und nächsten Keyframes berechnet wird. Aus dieser Vielzahl an Positionen aller Vertices setzt sich der aktuelle Keyframe, das momentane Bild, zusammen.

In Yage3D wird die Möglichkeit der Keyframe Animation unter dem Eigenschaftsfenster angeboten. Auf die genaue Handhabung wird später eingegangen. Vorweg Grundlegendes: Ein Keyframe wird durch die Klasse `Keyframe` angeboten. So ein Keyframe besitzt Vertices, um die Position im Raum zu definieren, Normalen, die unter anderem zur richtigen Beleuchtung des Objektes benutzt werden und Farbwerte. All diese Informationen müssen, um den Schein einer Bewegung zu simulieren, mit interpoliert werden. Die Interpolation übernimmt der Keyframe-Animator. Der Basis Keyframe Animtor befindet sich in der Engine. Zwei weitere, von ihm abgeleitete Animator Klassen befinden sich einmal im Runtime Modul, sowie im Editor. Der Keyframe Animator bietet weiterhin die Funktionen an, eine Animation zu starten, sie zu stoppen, und eine Animation zu wechseln. Die Animation wird durch die Klasse `KeyframeAnimation` realisiert. Eine Keyframe Animation enthält die Informationen über welche Keyframes eine Animation verfügt, genauer gesagt, aus welchen Bewegungssequenzen die Animation besteht. Diese Information speichert den Ort des Keyframes im Geo-Objekt als ganzzahligen Wert. Des weiteren wird die Zeit gespeichert, welche angibt, wieviel Zeit bis zum zugehörigen Keyframe vergehen soll. Ist die Zeit erreicht, so wird für die Animation auf den darauf folgenden Keyframe und seine Spielzeit referenziert. Wie oben schon erwähnt wird eine Keyframe Animation im dafür vorgesehenem Eigenschaftenfenster angelegt. Dieses Eigenschaftenfenster wird aus der Klasse `KeyframePropertiesPanel` erzeugt. Diese erzeugt eine tabellarische Übersicht über die gesamten Keyframes des zur Zeit ausgewählten Objektes. Außerdem bietet sie die Möglichkeit an dem Objekt neue Keyframes hinzuzufügen, bzw. vorhandene Keyframes zu entfernen. Nicht entfernt werden kann der Basis Keyframe. Über den Vertice-Modus lassen sich nun die einzelnen Keyframes der Schlüsselpositionen formen. Der gewünschte Keyframe sollte vorher in der Tabelle ausgewählt werden. Eine Animation kann auf zwei verschiedene Wege erstellt werden. Entweder wird sie erzeugt, indem man auf den dafür Vorhergesehen Button drückt, oder man benutzt das Kontex Menü, durch Betätigung der rechten Maustaste über einem Keyframe. In diesem Kontext Menü, sowie in dem Eigenschaftenfenster werden alle vorhandenen Animationen aufgelistet. Ein Keyframe wird einer Animation hinzugefügt,

indem er ausgewählt wird. Dann wird das Kontext-Menü geöffnet und der Menüeintrag „add to Animation: x“ ausgeführt. Es lassen sich die erstellten Animationen direkt bearbeiten indem man sie über ihre Schaltfläche auswählt. Der Name der ausgewählten Animation wird anschließend angezeigt. Sobald eine Animation ausgewählt wurde, wird die Keyframe Steuerleiste aktiviert. In ihr lassen sich alle der Animation zugehörigen Keyframes über den Schieberegler erreichen. Jetzt lassen sich auch die Abspielzeiten der Keyframes einstellen. Dazu wird der gewünschte Wert in das Zeit Fenster eingetagen. Der Editor bietet eine Vorschau der Animation an, damit eine Animation abgespielt werden kann, muss sie ausgewählt werden, anschließend wird der „Play“ Knopf gedrückt. Gestoppt wird die Vorführung über den „Stopp“ Knopf.

OH-Animation

Bei der Objekt-hierarchischen Animation (OH-Animation) werden im OHGeoObjekt bereits in der Welt befindliche Geometrie-Objekte hierarchisch animiert. Das bedeutet folgendes: Stellen wir uns den Arm eines Menschen vor. Dieser besteht, stark abstrahiert, aus drei Objekten, dem Oberarm, dem Unterarm und der Hand. Bewegt sich nun der Oberarm nach oben, bewegen sich auch der Unterarm sowie die Hand mit nach oben. Genau dieses realisiert die OH-Animation.

Jedes Geometrie-Objekt hat einen eigenen Knoten im Hierarchiebaum. Dieser Knoten wird durch die Klasse `OHNode` realisiert. In einem OHNode wird der Elternknoten sowie die Kind-Knoten - sofern vorhanden - gespeichert. So kann der Hierarchiebaum rekursiv aus OHNodes aufgebaut werden. In dem Beispiel mit dem menschlichen Arm würde der Oberarm, der Unterarm und die Hand jeweils einen Knoten repräsentieren. Der Unterarm wäre ein Kindknoten vom Oberarm, die Hand wäre ein Kindknoten vom Unterarm. Der Handknoten besitzt keine Kinder. Somit ist der Oberarm der Elternknoten vom Unterarm, der wiederum Elternknoten der Hand ist.

Die Geometrie-Objekte im OHGeoObjekt besitzen einen Verbindungspunkt, den sogenannten `JointPoint`. Dieser JointPoint beinhaltet Informationen über die Rosition und die Rotation des Geometrie-Objekts *relativ* zu seinem Elternknoten. Ein JointPoint ist also kein richtiger Punkt, allerdings wird er im Editor als ein Punkt dargestellt. Der JointPoint befindet sich standardmäßig am Ursprung des Geometrie-Objekts. Intern liegt er immer am Ursprung des lokalen Koordinatensystems, wenn der JointPoint innerhalb des Geometrie-Objekts verschoben wird, verschiebt sich intern auch das lokale Koordinatensystem. Davon bekommt der Benutzer allerdings nichts mit.

Auch bei dieser Animation gibt es Keyframes, die `OHKeyframes`. In ihnen wird, im Gegensatz zur Keyframe-Animation, nur die JointPoints der Geometrie-Objekte gespeichert. Auch hier wird eine Zeitangabe (in Millisekunden) mitgespeichert, welche die Zeit repräsentiert, die zwischen diesem und dem nächsten OHKeyframe bergehen soll.

Fertige Animationen werden in `Animation` gespeichert. Ein Animationsobjekt besitzt einen Namen, damit der Benutzer die Animation im Animationsbrowser (Abschnitt 2.3.8) identifizieren kann. Weiterhin besitzt eine Animation eine Liste mit Referenzen auf die in ihr enthaltenen OHKeyframes inklusive deren Zeitangaben. Zusätzlich kann festgelegt werden, ob und wie oft sich diese Animation wiederholen soll und in welchem Durchlauf

sie sich befindet.

Für den Ablauf der Animation ist der `OHAnimator` zuständig. Dieser verhält sich genauso wie der GeoAnimator.

Zu jedem OHGeoObject gehört ein OHAnimator, der die Interpolation zwischen den OHKeyframes übernimmt. Der OHAnimator ist zeitgesteuert, weshalb er das komplette Zeitmanagement der OH-Animation übernimmt. Er kennt alle für das OHGeoObject erstellten Animationen, wobei die letzte, die nächste sowie die default-Animation extra vermerkt sind. Der OHAnimator ist in der Lage, zu jeder Zeit in eine andere Animation zu wechseln.

Zur graphischen Visualisierung der JointPoints im OH-Animationsmodus des Editors wird noch eine weitere Zeichenebene, der `OHOverlayDrawer` zur Verfügung gestellt. Hierauf werden die JointPoints sowie Verbindungslinien zwischen den hierarchisch zusammengehörenden JointPoints gezeuchent. Ebenso werden, bei selektierten Objekten, auch die Selektionslinien auf dieser Ebene gezeichnet.

lightobject

In dem Paket `yage3d.engine.objects.lightobject` befindet sich das Lichtobjekt.

In dem `LightObject` sind Informationen über die Farbwerte der Lichtquelle gespeichert sowie Werte zur Abschwächung des Lichts. Ebenso sind hier Spotlight-spezifische Attribute vorhanden, so dass es diesem Licht-Objekt egal ist, welcher Art es angehört. Alle Typen von Lichtquellen, die im Editor vorhanden sind, können also von diesem Objekt repräsentiert werden. Desweiteren besitzt das LightObject Methoden, mit denen man Lichtparameter ändern und das „Licht“ einschalten kann.

materialobjects

Das Paket `yage3d.engine.geoobjects.materialobjects` übernimmt die Materialienverwaltung in der Engine. Dazu notwendig ist die Klasse `Material`, die ein Material in der Engine repräsentiert. Diese Klasse managed alles, was man für ein Material braucht, das heißt bestimmte Materialeigenschaften, Texturen und Colormapping. Die Klasse `MultiTextureHelper` erleichtert den Umgang mit Multitexturing.

particleobjects

Dieses Paket, `yage3d.engine.geoobjects.particleobjects`, beinhaltet drei verschiedene Klassen, die zur Realisierung der Partikel-Objekte notwendig sind. In `ParticleData` werden die Partikel-Eigenschaften definiert. Dazu gehören Angaben wie Anzahl der Partikel, das Aussehen, verschiedene Geschwindigkeitsangaben und das Gravitationsverhalten. Ein `ParticleObject` besitzt eine Liste `Particle` und Partikel-Eigenschaften (`ParticleData`). Partikel enthalten Attribute über Farbwerte, Positions-

, Richtungs- und Gravitationsangaben sowie eine Lebensdauer. In dem Partikel-Objekt werden die einzelnen Partikel initialisiert und gezeichnet. Der Ablauf wird durch die update-Methode gesteuert, durch die das Partikel-Objekt aktualisiert wird. Die Partikel können mit Texturen belegt werden, dazu wird die Klasse `TextureLoader` benötigt. Diese Klasse stammt aus einem Tutorial [8] und wurde von uns angepasst.

3.2.2 Engine-Objekts, die nicht im Editor-Teil zu finden sind

Es gibt Objekte, die keine Erweiterungen für den Gebrauch im Editor benötigen, daher sind diese Objekte nur in der Engine zu finden. Dazu gehören Sound-Objekte, die mittels Trigger in der Welt positioniert und abgespielt werden können. Ebenso Nebelobjekte und die sogenannte Skybox sind nur in der Engine zu finden.

soundobjects

Das Paket `yage3d.engine.geoobjects.soundobjects` enthält zwei Klassen zur Soundverwaltung. `SoundObject` repräsentiert ein Sound-Objekt in der Engine und `Listener` stellt den Hörer für ein solches Sound-Objekt zur Verfügung. Ein Sound-Objekt enthält eine Sound-Datei als Soundquelle mit der Endung .wav sowie Attribute zum Verwalten und den Status der Soundquelle. Soundquellen werden mit Hilfe der OpenAL-Programmbibliothek verwaltet, daher besitzt ein Sound-Objekt auch die notwendigen OpenAL-Attritbute (Abschnitt 5.4).

Der Listener enthält Positionsangaben und weitere Attribute, die zum Hören einer Soundquelle wichtig sind. Der Sound wird relativ zur Entfernung des Hörers lauter oder leiser bzw. aus gestellt. Die dafür benötigten Angaben ergeben sich aus der Position des Listeners sowie den Attributen von der Soundquelle. Diese Berechnungen erledigt OpenAL.

fogobject

In dem Paket `yage3d.engine.geoobjects.fogobject` gibt es nur eine Klasse, das `FogObject` (Nebelobjekt). Das Nebelobjekt repräsentiert Nebel in der Welt. Dazu benötigt das Nebelobjekt Informationen über den Nebelmodus, die Ausdehnung des Nebels und dessen Farbe. Als Modus stehen drei verschiedene Modi zur Verfügung, EXP, EXP2 und LINEAR.

EXP ist nicht sehr rechenintensiv und nicht sehr schön, daher ist dieser Modus eher für ältere Computer geeignet. EXP2 lässt den Nebel realistischer Aussehen als mit EXP, ist dafür aber auch rechenintensiver. LINEAR, im Nebelobjekt standardmäßig eingestellt, ist der Modus, der den Nebel am Besten aussehen lässt, dafür ist er auch der Modus mit der höchsten Rechenintensität.

Wie alle Geometrie-Objekte ist auch der Nebel in der Lage, sich selbst zu zeichnen.

skyobject

Das Paket `yage3d.engine.geoobjects.skyobject` besteht, wie das Nebelobjekt, aus einer einzelnen Klasse. Das Skyobjekt (`SkyObject`), in unserem Fall eine Sky-Box, bildet einen Quader um die komplette Welt. In diesem Quader können Texturen für den Himmel sowie für den Boden angegeben werden. Die Sky-Box wird im Editor nur in der Perspektiv-Ansicht gezeichnet. Man könnte alternativ auch die Box durch eine Sphere (Kugel) ersetzen, was aber nicht im Editor realisert ist.

textureobjects

Für das Material benutzbare Texturen, werden in dem Paket `yage3d.engine.geoobjects.textureobjects` realisiert. Das `NewTextureObject` enthält Attribute zur Verwaltung einer Textur mit OpenGL. Mit Hilfe des `TextureLoader` wird das Laden der Texturen realisiert. Diese Klasse lädt das Bild von der Festplatte in einen Puffer, mit dem dann gearbeitet werden kann.

3.3 projectmanagement

Das Projektmanagement verwaltet die vom Benutzer im Editor angelegten Projekte. Verwaltung bedeutet in der Engine die Speicherung der Projekt-Daten sowie die Bereitstellung der Speichermethoden für die Objekte, die im Editor erstellt werden, sowie die Bereitstellung einer Projektstruktur.

Die Projektstruktur befindet sich in der Klasse `ProjectDescription`. Hier wird die Struktur des Projektordners mitsamt Unterordnern festgelegt.

In dem Paket `yage3d.engine.projectmanagement` befinden sich die Klassen, die zur Speicherung der Projektdaten und Objekten in XML-Dokumente notwendig sind. Dafür werden verschiedene Schreib- und Lese-Klassen bereitgestellt. Weiterhin wird ein `XMLObject` benötigt. Ein XML-Objekt verwaltet intern ein org.w3c.dom.Element und bietet Funktionen, welche die XML-Serialisierung von XMLWriteable-Objekten erleichtert.

Speicherung

Zum Speichern, also das Schreiben in eine XML-Datei, wird der `XMLWriter` benötigt. Der XML-Writer realisiert die Speicherung von Welten und deren Inhalt in XML. Damit dieses funktioniert, wird ein `XMLFileWriteable`-Objekt benötigt, welches von `XMLWriteable` abgeleitet ist. XMLWriteable stellt zwei Methoden, die zur Speicherung essentiell sind, zur Verfügung. Die eine Methode liefert das passende XML-Tag des Objekts zurück und die andere bereitet die Speicherung des Objekts vor.

Zur Speicherung wird ebenfalls ein `XMLFileWriteable`-Objekt benötigt. XMLFileWritea-

ble muss von allen zu speichernden Objekten implementiert werden, denn dieses Interface enthält Methoden, mit denen man den Speicherort der Objekte abfragen kann.

Bei der Speicherung der Welt in eine XML-Datei wird ein XMLFileWritable-Objekt mitsamt dessen Kindern gespeichert. Falls eines oder mehrere der Kinder ebenfalls XMLFileWriteable-Objekte sind, dann werden diese in separaten Dateien abgelegt. Kennt ein XMLFileWriteable bereits seinen Dateinamen, wird gegebenenfalls die bereits bestehende Datei überschrieben. Anderenfalls wird dem Objekt ein noch freier Dateiname zugeordnet und per setFileName() mitgeteilt. Jedes XMLFileWriteable-Kind wird nur einmal gespeichert, auch wenn es an mehreren Stellen referenziert sein sollte.

Weiterhin wird die Klasse `XMLString` benötigt, welche die Namen der XML-Tags aller Engine-Klassen enthält. Dadurch wird eine einheitliche Struktur geschaffen und die XML-Dateien werden leserlicher. Der `XMLHelper` legt die Struktur der XML-Dateien durch Setzen bestimmter Tags fest.

Auslesen

Zum Auslesen der XML-Dateien wird ein Reader benötigt. Die abstrakte Klasse `AbstractXMLReader` stellt eine komfortable Schnittstelle zu einem SAX-Parser dar. Er dient als Oberklasse für den `XMLReaderScene` und für weitere Reader, die nicht Teil der Engine sind. Der `XMLReaderScene` ist für das Laden der Welten und dessen Inhalt zuständig. Dazu werden die XML-Dateien ausgelesen und die Welt in den Editor geladen. In diesem Paket gibt es eine weitere abstrakte Klasse, das `AbstractXMLReaderObject`. Diese Klasse ist Oberklasse für einen XMLReader, der Geometrie-Objekte aus Dateien instanziiert. Der AbstractXMLReader bringt bereits Funktionalität mit sich, die verhindert, dass eine Datei zweimal instanziiert wird.

3.4 Scenelogic

Das Paket `yage3d.engine.scenelogic` stellt verschiedene Klassen für die gesamte Logik der Scene zur Verfügung. Hier befinden sich die verschiedenen Knotentypen, die im Szenenbaum vorkommen, sowie zum Anzeigen benötigte Klassen. Weiterhin befinden sich in diesem Paket auch Klassen und Methoden zur Manipulation.

3.4.1 Knotentypen

Es gibt verschiedene Knotentypen, die in den Szenenbaum eingehängt werden können:

- `BaseNode`, der Basisknoten
- `GroupNode`, der Gruppenknoten
- `WorldNode`, der Weltknoten

- `ObjectNode`, der Objektknoten
- `LightNode`, der Lichtknoten
- `ParticleNode`, der Partikelknoten
- `TriggerNode`, der Triggerknoten

`BaseNode` ist die Basisklasse für sämtliche Knotentypen, die sich im Szenenbaum befinden können, um verschiedene Objekte in der Welt hierarchisch zu platzieren und zu gruppieren. Jeder Knoten hat einen eindeutigen Namen und umfasst weiterhin Informationen über den Knotentyp, etwaige Transformationen der Objekte in dem Knoten und dessen BoundingVolume (Abschnitt 3.1.1). Ebenso weiß jeder Knoten, wer sein Elternknoten ist, ob er gezeichnet wird und ob er in der aktuell angezeigten Welt vorhanden ist. Jeder Knotentyp ist in einem XML-Dokument speicherbar, die dafür benötigten Methoden müssen von jedem Knotentyp implementiert werden. Weiterhin ist jeder Knotentyp in der Lage, sich selbst, bzw. die in ihm liegenden Objekte zu zeichnen.

`GroupNode` ist eine Klasse zur Realisierung eines Gruppenknoten im Szenenbaum. Charakteristisch für Gruppenknoten ist, dass sie Kinder besitzen. Diese Kinder können wiederum Gruppenknoten sein oder auch Blätter, welche aus verschiedenen Knotentypen bestehen können. Blätter besitzen keinerlei Kind-Knoten.

`WorldNode` ist ein spezieller Gruppenknoten und dient zur Realisierung des Wurzelknotens der Welt. Der Wurzelknoten besitzt keinen Elternknoten, da der Wurzelknoten den Ursprung des Szenenbaums bildet. Im Weltknoten werden spezielle Weltdaten, beispielsweise der zugehörige Dateiname und in der Welt vorhandene Texturen festgehalten.

`ObjectNode` ist eine Klasse zur Realisierung von Objektknoten, die Geometrie-Objekte enthalten, welche mit Hilfe der Objektknoten in den Szenenbaum eingehängt werden können. Im Objektknoten können nur Geometrie-Objekte vorhanden sein, andere Objekte, beispielsweise Licht- oder Trigger-Objekte, sind in ihnen nicht speicherbar.

`LightNode` ist eine Klasse zur Realisierung von Lichtknoten, die Lichtquellen enthalten, welche mit Hilfe der Lichtknoten in den Szenenbaum eingehängt werden können. Im Lichtknoten können nur Licht-Objekte vorhanden sein, andere Objekte sind in ihnen nicht speicherbar. Weiterhin kann es maximal nur 8 verschiedene Lichtknoten und Lichtobjekte geben, da diese Anzahl von OpenGL eingegrenzt und festgelegt wird. In einer Welt gibt es mindestens eine Lichtquelle, da sonst auf dem Bildschirm nichts zu erkennen wäre.

`ParticleNode` ist eine Klasse zur Realisierung von Partikelknoten, die Partikel-Objekte enthalten, welche mit Hilfe der Partikelknoten in den Szenenbaum eingehängt werden können. Im Partikelknoten können nur Partikel-Objekte vorhanden sein, andere Objekte, beispielsweise Licht- oder Trigger-Objekte, sind in ihnen nicht speicherbar.

`TriggertNode` ist eine Klasse zur Realisierung von Triggerknoten, die Trigger-Objekte enthalten, welche mit Hilfe der TriggerNodes in den Szenenbaum eingehängt werden können. Im Triggerknoten können nur Trigger-Objekte vorhanden sein, andere Objekte, beispielsweise Licht- oder Geometrie-Objekte, sind in ihnen nicht speicherbar.

3.4.2 Darstellung der Welt

In dem Paket `yage3d.engine.scenelogic` befinden sich auch Klassen, die zur Darstellung der Welt, oder auch der Szene, zuständig sind. Die Klasse `Camera` realisiert ein Kamera-Objekt. Jede der vier verschiedenen Ansichten im Normal-Modus des Editors ist mit einer eigenen Kamera verbunden, um die verschiedenen Sichtweisen auf die Objekte in der Szene bereitzustellen. Die Kamera benutzt ein Frustum-Culling, das angibt, welche Objekte in der Welt potentiell sichtbar sind und demnach gezeichnet werden. Objekte, die nicht im Frustum liegen, werden somit auch nicht gezeichnet. Grundlage dafür ist der hierarchische Aufbau der Szene. Jeder Knoten besitzt eine Bounding-Volume die ihn und seine Kinder einschließt. Die Berechnung dieser hierarchischen Bounding-Volumes geschieht in einem rekursivem Durchgang, der nach der Bewegung eines Objektes angestoßen wird. Für das Culling wird ein Test der Bounding-Volume mit den 6 Ebenen des Frustums durchgeführt. Gewissermaßen als Abfallprodukt fällt durch den Test der Near-Ebene der Abstandwert dieses Knotens an. DIe Genauigkeitdes Wertes hängt zwar von der Ausdehnung des Knotens ab, ist jedoch gut genug, um zukünftig für Level-of-Detail genutz werden zu können.

Die Szene, bzw. der Szenenbaum, wird durch die Klasse `Scene` realisiert. Diese Szene kann maximal vier Ansichten erstellen und diese zeichnen, davon ist eine Ansicht die 3D- oder auch Perspektivenansicht, die sowohl von Editor, als auch vom Runtime-Modul genutzt wird. Die verbleibenden drei Ansichten sind 2D- beziehungsweise orthogonale Ansichten, welche jeweils die selbe Welt von verschiedenen Seiten zeigen. Um dies zu realisieren, besitzt die Szene entsprechend viele Kameras und GLEventListener, die auf den gleichen Welt-Daten operieren.

Der von OpenGL angebotene GLEventListener wird durch `SceneGLEventListener` bereitgestellt. Er reagiert auf Veränderungen der Szene und zeichnet anschließend die Szene neu.

Ebenso wird in diesem Paket ein angepasster Zeichner, der `CustomDrawer`, realisiert. `CustomDrawer` ist ein Interface, welches für die erbenden Klassen eine Zeichenmethode vorschreibt. Die Klassen `yage3d.engine.geoobjects.OHOverlayDrawer` und `yage3d.editor.controller.OverlayDrawer` sind von diesem Interface abgeleitet. Diese Klassen bereiten eine Art weitere Zeichenebene an, auf der u.A. die Selektionslinien gezeichnet werden.

Damit überhaupt selektiert (picking) werden kann, wird eine Klasse benötigt, die diese Selektion bereitstellt und verwaltet. Der `PickingOperator` ist dafür zuständig. Er ist eine Art Monitor, der Objekt-Selektion in der Szene beobachtet und ausführt. Dabei überwcht er verschiedene Objekt-Typen, die per Maus selektierbar sind. Dafür muss der PickingOperator zuvor gestartet werden und sich in dem Arbeitsmodus befinden. Es gibt drei verschiedene Arbeitszustände:

- idle - der PickigOperator befindet sich im Warte-Modus
- ready - der PickingOperator ist bereit
- working - der PickingOperator arbeitet bereits

Es können mit Hilfe dieses Monitors verschiedene Objekt-Typen selektiert werden: Das komplette Geometrie-Objekt, einzelne Faces oder Vertices. Dabei beobachtet der Monitor nur die Objekte, der Modus wird im Editor festgelegt. Es werden die beobachteten Objekte sowie die selektierten Objekte im Monitor gespeichert, wodurch gewährleistet wird, dass nur beobachtete Objekte selektiert werden können. Zudem können mehrere Objekte gleichzeitig selektiert werden. Ebenso reagiert der PickingOperator auf selektierte Knoten im Szenebaum bzw. in der Objekthierarchie, wodurch auch die selektierten Knoten im Monitor gespeichert werden.

Zur Darstellung der Welt gehört auch die Bereitstellung von Geräuschen, realisiert durch die Klasse `Sound`. Sie übernimmt die Verwaltung von Sound-Objekten mit Hilfe von OpenAL. Sie sorgt dafür, dass eine Verbindung zur Soundkarte hergestellt wird, so dass auch Sounds in die Welt eingefügt werden können und die Engine diesen auch verarbeiten kann.

3.4.3 Transformationshilfen

Die Objekte in der Welt können durch unterschiedliche Operationen transformiert werden. Um diese Transformationen zu erleichtern, bietet das Paket Klassen an, die die Verwaltung der Transformationen übernimmt.

`Transformation` Transformationen wie Translation, Skalierung und Rotation für Objekte. Die 4x4 Transformationsmatrix, der 3-stellige Translationsvektor sowie die 3x3 Rotationsmatrix werden in dieser Klasse gespeichert. Werden die Objekte transformiert, werden die Änderungen vorerst in der Transformationsmatrx gespeichert und dann an OpenGL übergeben. Somit sind spezifische Transformationen speicherbar und immer abrufbereit.

Die Berechnung der Transformationen unterstützen zwei für unseren Gebrauch angepasste Klassen, einmal die `Matrix3f` und `Vector3f`.

Die Matrix ist eine 3x4-Matrix als einzeiliger Vektor. Diese Matrix darf nur float-Werte enthaöten, was der Name 3f schon andeutet. Sie realisiert die Rotationsmatrix mit angehämgtem Translationsvektor, also ist sie sogesehen die nicht homogenisierte Form der Transformationsmatrix. Diese Klasse führt Multiplikationen auf Matrizen durch, also die Multiplikation mit einer Matrix oder einem dreistelligen Vektor.

Dieser Vektor ist vom Typ `Vector3f`, ein dreistelliger Zeilenvektor. Diese Klasse stellt Vektoradditionsmethoden zur Verfügung, so dass mehrere Vektoren addiert werden können.

4 Runtime Modul

In diesem Kapitel wird das Runtime-Modul beschrieben.

Die Model-Schicht wird durch die Objekte aus dem Paket `yage3d.runtime.objects.runtimeobjects` repräsentiert. Die View besteht aus dem Paket `yage3d.runtime.gui` und der Controller wird durch das Paket `yage3d.runtime.controller` realisiert. Über diesen Paketen steht, wie auch im Editor, die `RuntimeMain`-Klasse, durch die das Runtime-Modul gestartet wird.

4.1 Model

Der Model-Bereich des Runtime Moduls besteht auch hier aus Objekten, die in dem Paket `yage3d.runtime.objects.runtimeobjects` zu finden sind. Hier finden sich die GeoObjekte, Materialien, Lichter, Animatoren, sowie der angepasste WorldNode.

4.1.1 Geometrie-Objekte

Die Geometrie-Objekte sind auf den Gebrauch im Runtime-Modul zugeschnitten, da hier nicht alle Attribute und Methoden der Geometrie-Objekte benötigt werden, die im Editor-Teil notwendig sind. So enthalten die Runtimeobjekte zB. keine Surfaces mehr, sondern arbeiten allein auf Mengen von Primitien.

Das `RTBaseGeo` definiert einen Abstrakten Grundtyp eines Geometrie-Objekts. Dessen Inhalte werden vom `RTGeoStatic` implementiert. Dieses enthält ein ein RTDummyobject, daß die Objektdaten in Form von Vertices und Primitivn enthält, sowie die Möglichkeit, diese Daten auch in einer Displayliste zu sichern. (Sobald dies geschieht, wird der Dummy überflüssig.)

Das `RTKeyframeAnimatedObject` und das `RTOHGeo` realisieren die jeweiligen Animationsstypen analog zum GeoStatic.

4.1.2 Material

Die Geometrie-Objekte des Runtime-Moduls besitzen Materialien. Diese realisiert die Klasse `RTMaterial`. Während die Materialklasse des Editors dafür zuständig ist, das

Material pro Surface zu verwalten, verwaltet das RTMaterial Daten pro Face; funktioniert ansonst analog.

4.1.3 Licht

Das `RTLight` entspricht in seiner Funktionsweise dem Lichtobjekts des Editors, wobei die Funktionen für das Anzeigen des Licht-Preview-Objekt nicht enthalten sind.

4.2 View

Der View-Bereich des Runtime Moduls besteht, wie auch im Editor, aus einem Graphical User Interface (GUI). Die dafür benötigten Klassen befinden sich in dem Paket `yage3d.runtime.gui`.

In dem Paket befindet sich die Klasse `MainFrame`, die das Hauptfenster des Runtime Moduls generiert. Eingaben in diesem Fenster werden von dem `RuntimeMouseHandler` verwaltet.

Das Hauptfenster ist ein einfacher JFrame, in dem direkt der GLCanvas sitzt. Das QuitEvent des Fensters wird an den EventController übergegeben. Maus-Aktivitäten in diesem Fenster werden vom Handler abgefangen, der die entsprechenden Methoden ausführt.

4.3 Controller

Der Controller-Bereich des Runtime Moduls, zu finden im Paket `yage3d.runtime.controller`, stellt die für das Runtime Modul notwendigen Kontroll-Klassen zur Verfügung.

Alle im Paket befindlichen Controller werden im `RuntimeMainController` angemeldet und verwaltet. Dieses ist eine Singleton-Klasse, das heißt, es darf jederzeit maximal ein MainController-Objekt existieren. Über diesen MainController werden die weiteren Controller angelegt und verwaltet. Der Zugriff auf diese verschiedenen Controller geschieht ausschließlich über den MainController.

Damit im Hauptfenster des Runtime-Moduls etwas sichtbar ist, muss eine Szene angelegt werden. Das Szene-Objekt befindet sich in diesem Paket in der Klasse `RuntimeScene`. Hier wird der Animator und eine eigene Kamera angemeldet. Außerdem ist die RuntimeScene in der Lage, sich selbst, d.h. alle Objekte in der Szene, zu zeichnen. Dazu wird ein GLEventListener von OpenGL benötigt, um OpenGL-Methoden auszuführen. Dieser Listener wird durch die Klasse `RuntimeGLEventListsner` bereitgestellt.

Der `EngineController` dient der Steuerung aller Engine-bezogenen Aktionen. Das sind

Level-Lade-Operationen, Änderungen der Bildschirmauflösung und Vollbild-Einstellungen.

Das `FrameLogicInterface` stellt dem Programmierer ein Interface bereit, um den Ablauf der Framedarstellung mit eigenem Code zu beschreiben. Hier kann er auf Benutzereingaben reagieren und Dinge wie Overlays zeichnen. Die `DefauktFrameLogic` stellt eine solche vordefinierte Klasse dar, auf die zurückgegriffen wird, wenn der Programmierer keine eigene angegeben hat.

Innerhalb der FrameLogic kann der Programmierer auf Funktionen der `YageAPI` zurückgreifen. Die in dieser Klasse definierte Schnittstelle bietet Funktionen zur Objektmanipulation, zur Abfrage von Eingaben, zur Ausgabe von Text, und zur Kollisionserkennung.

5 Externe Bibliotheken

Im folgenden werden die im Yage-Projekt benutzten externen Bibliotheken beschrieben.

5.1 Java bindings for OpenGL (JOGL)

JOGL bringt OpenGL für Java. Dies geschieht mit Hilfe von JNI (Java Native Interface). Genauer gesagt werden alle JOGL Befehle per JNI auf die entsprechenden C-Befehle von OpenGL gemapped. Dadurch ist es möglich mit Java zu programmieren und dennoch OpenGL-Befehle zu benutzen. Der größte Vorteil an JOGL ist, dass es auf fast allen Plattformen läuft. Damit kann man plattformunabhängige OpenGL-Anwendungen schreiben, da Java ebenfalls sehr unabhängig ist.

JOGL stellt (fast) alle Features von OpenGL zu Verfügung. Neben der GL-Klasse also auch die GLU und die GLut.

JOGL bietet noch weitere Features, wie einen Animator, der die Zeichenfläche (Canvas) immer wieder neu zeichnet. Aufgrund einiger Probleme haben wir aber einen eigenen Animator geschrieben, der besser zu unserem Editor gepasst hat.

5.2 Hilfe-Dateien

Wir haben für unser Hilfesystem JavaHelp 2.0 eingesetzt, Das JavaHelp API, das von SUN entwickelt wurde, definiert einen Standard für ein XML/HTML basiertes Hilfe-System. Dieses System kann in jedem Browser und auf allen Plattformen, für die es eine Java-Virtual-Machine gibt, angezeigt werden.

Die wesentlichen Bestandteilen von JavaHelp sind:

- **HelpSet**

  Ein HelpSet besteht aus einigen XML-Dateien, mit der die Struktur beschrieben werden, dem Index und den HTML-Seiten mit dem eigentlichen Inhalt. Sie können wieder untergegliedert werden.

  - **Organisations-/Navigationsdateien**
    Die Datei kümmert sich um die Pfadangaben zu Map.jhm, Index. xml und TOC.xml und gilt als Anker im Hilfesystem. Sie wird als Initialdatei vom Hilfesystem benötigt.

Map.jhm
In dieser Datei wird eine Zuodnung von Help-IDs zu echten Pfadangaben vorgenommen.

Index.xml
wird definiert, mit welchem Bezeichner eine Hilfeinformationsdateid im Hilfefenster erscheinen soll.

TOC.xml
Die Datei TOC.xml beschreibt die Struktur, das Inhaltsverzeichnis (TOC = Table Of Contents) für das Hilfefenster.

– **Hilfeinformationsdateien**
Sie handelt sich um herkömmliche HTML-Dateien. Die Ansteuerung der Hilfeinformationsdateien in JavaHelp erfolgt, indem der genaue Bezeichner für die Hilfedatei aufgerufen wird. In dieser Realisierung des Hilfesystems entspricht die HelpID des Hilfesystems genau dem Bezeichner in JavaHelp, der die Hilfeinformationstexte identifiziert. (In JavaHelp wird dieser Bezeichner MapID genannt.)

- **Helpviewer**
  Die Hilfe wird mit dem HelpViewer angezeigt. HelpViewer sind sowohl als Standalone-Komponente (Swing) als auch Browser-basiert (Applet) und Server-basiert (JSP-Seiten) verfügbar. HelpSets können lokal oder im Netzwerk liegen.

- **HelpBroker**
  Der HelpBroker ist ein Agent, der zwischen der Applikation und dem HelpViewer vermittelt. Weiterhin stellt er Methoden zur Verfügung, die benutzt werden können, um kontext-sensitive Hilfe zu realisieren. Innerhalb der Applikation wird die HelpTopicID für graphische Objekte oder einen bestimmten Kontext festgelegt und verweist auf Inhalt im HelpSet.

5.3 Jython

hier steht text...

5.4 Java bindings for OpenAL (JOAL)

Wir haben uns für JOAL entschieden, da es für eine realistische, akkustische Darstellung einer Welt die optimalen Vorraussetzungen besitzt. Dies vorallem deswegen, da es 3D Sound unterstützt. Auch die gute Portabilität und die JOGL-ähnliche Syntax sprachen dafür.

In der Anwendung ist JOAL verhältnismäßig einfach zu handhaben:

Es im Grunde drei wichtige Aspekte zu beachten. Diese sind „Source“, „Buffer“ und „Listener“. Mit diesen lässt sich in OpenAL schon sehr viel machen. Mit Java kann man sehr

gut objectorientiert mit diesen „Teilen“ arbeiten. Mit der *ALut* (Audio Library Utilities) kann man ohne viel Code eine „.wav“ Datei laden, was wir auch nutzen.

Source

Eine Source ist, wie der Name schon vermuten lässt, eine Sound-“Quelle“. Diese werden mit Sounddaten aus einem oder mehreren *Buffern* gefüttert und können diese dann abspielen. Die Sourcen lassen sich frei im 3D-Raum positionieren. Die eigentliche Berechnung von Lautstärke, Dopplereffekt oder ähnlichem erfolgt erst bei korrektem Einstatz eines *Listeners*. Man kann mit den Sourcen noch viele weitere Dinge steuern. Insbesondere sollte auch der *RelativeToListener*- Parameter nicht übersehen werden, da dieser den 3D Sound für eine Source ausstellen kann. Die Position dieser Source würde sich immer mit dem Listener mitbewegen also keine Änderung in der Lautstärke erhalten.

Listener

Der Listener repräsentiert den Hörer (also den Spieler). Es gibt immer nur exakt einen Hörer. Wenn dieser sich bewegt berechnet OpenAL je nach Umgebungsparametern das akkustische Signal der Sourcen. Ohne einen explizit gesetzen Listener ist kein 3D Sound zu erwarten.

Buffer

Wie schon geschildert, werden die Audiodaten in Buffern an die Sourcen übergeben. Die OpenAL Buffer arbeiten dabei mit *ByteBuffer*n. Es ist möglich mehrere Buffer an einer Source zu *queuen*, also nacheinander abspielen zu lassen. Dies würde insbesondere bei Hintergrundsmusik Sinn machen.

6 Links

[1] http://www.volanakis.de/wfp-folien.pdf (2006-03-04)

[2] http://www.iwr.uni-heidelberg.de/groups/comopt/teaching/uml/html/kapitel_3_7_1024x768/sld025.htm

[3] http://ag-kastens.uni-paderborn.de/lehre/material/seminar_refactoring/semref-7-folien.pdf

[4] http://www1.informatik.uni-wuerzburg.de/database/courses/pi2_ss03_dir/Entwurfsmuster_2.pdf

[5] http://www.dpunkt.de/java/Programmieren_mit_Java/Oberflaechenprogrammierung/40.html

[6] http://www.schule.de/schulen/oszhdl/gymnasium/faecher/informatik/ooa-ood/mvc_0.htm

[7] http://userpage.fu-berlin.de/ ram/pub/pub_jf47htdHt/mvc

[8] http://pepijn.fab4.be/nehe/lesson19.jar