Gasch, Robert

Entwurf und prototypische Implementierung eines Systems zur Erstellung von 3D-Programmen durch Modellierung eines Szenegraphen und automatischer Quelltextgenerierung

Diplomarbeit

Hochschule Mittweida

University of Applied Sciences

Fachbereich Mathematik/Physik/Informatik

Mittweida, 2009

Gasch, Robert

Entwurf und prototypische Implementierung eines Systems zur Erstellung von 3D-Programmen durch Modellierung eines Szenegraphen und automatischer Quelltextgenerierung

eingereicht als

Diplomarbeit

an der

Hochschule Mittweida

University of Applied Sciences

Fachbereich Mathematik/Physik/Informatik

Mittweida, 2009

Erstprüfer: Prof. Dr.-Ing. Rainer Gaudlitz

Zweitprüfer: Prof. Dr. rer. nat. Konrad Schulz

Vorgelegte Arbeit wurde verteidigt am: 02.10.2009

Bibliographische Beschreibung:

Gasch, Robert:
Entwurf und prototypische Implementierung eines Systems zur Erstellung von 3D-Programmen durch Modellierung eines Szenegraphen und automatischer Quelltextgenerierung. - 2009 - 87 S.
Mittweida, Hochschule Mittweida, Fachbereich Mathematik / Physik / Informatik, Diplomarbeit, 2009

Referat:

Ziel der Diplomarbeit war es, ein Programm zur automatischen Erstellung von 3D-Programmen zu entwickeln. Es bietet dem Benutzer einen einfachen Einstieg in die Welt der 3D-Programmierung. Beginnend mit der Modellierung eines Szenegraphen bis zum fertigen Programm unterstützt es ihn auf vielfältige Weise.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingsverzeichnis

Tabellenverzeichnis

Glossar

Element

Ein Element ist ein Knoten oder eine Knotenkomponente

JDK

Java Development Kit, eine Entwicklungsumgebung für Java

JRE

Java Runtime Environment, eine Laufzeitumgebung für Java-Programme

Komponente

Ein Knoten, eine Knotenkomponente oder eine Kante

System

Als System wird das zu entwickelnde Programm bezeichnet

1. Einführung

1.1. Charakteristik des geplanten Codegenerators

Im Rahmen dieser Arbeit wird ein Programm entwickelt, das dem Benutzer das Erstellen einer 3D-Anwendung erleichtert. Dazu modelliert er einen Szenegraph, aus welchem dann ein bereits lauffähiges Gerüst einer 3D-Anwendung erstellt wird. Dabei muss der Benutzer lediglich grundlegendes Wissen über den Aufbau eines Szenegraphen besitzen.

Die Idee zu dieser Arbeit entstand während der Erarbeitung eines Belegs im Fach "Grafiksysteme". Bei der Entwicklung wäre ein solcher Codegenerator sehr nützlich gewesen.

Die Anwendung soll folgende Merkmale besitzen:

- Kernmerkmale:
  - Modellierung eines Szenegraphen:
    - prototypische Unterstützung von Java3D
    - Überprüfung der Semantik des Szenegraphen und Verhinderung von illegalen Relationen
  - Quellcodeerzeugung:
    - Erzeugung von einem lauffähigem Java-Quellcode-Gerüst mit übersichtlicher Formatierung, welches durch individuelle Anpassungen erweitert werden kann
    - Nutzung von Kommentaren

  - Oberfläche:
    - übersichtliche Einteilung in funktionale Gruppen
    - Beschriftung mit standardisierten Symbolen
    - Beschriftung in der jeweiligen Landessprache (Deutsch und Englisch)
  - Speicherung eines Szenegraphen in einer Datei in Form von serialisierten Objekten
  - modulares Design zur späteren Unterstützung mehrerer 3D-Standards
- Optionale Merkmale:
  - Speicherung eines Szenegraphen als XML (Extensible Markup Language)
  - Exportieren eines Szenegraphen als PNG (Portable Network Graphics)
  - Drucken eines Szenegraphen

Mit Java3D und JOGL existieren zwei verschiedene Erweiterungen, um mit Java ein 3D-Programm zu erstellen. Deshalb wird der Codegenerator modular aufgebaut. In Zukunft kann damit für beide Varianten Quellcode erstellt werden. Der Unterschied zwischen beiden ist die Abstraktionsebene. Java3D bietet ein objektorientiertes Programmiermodell mit einem Szenegraphen, welcher den Vorteil eines schnellen Einstiegs bietet. Nachteilig ist, dass keine manuellen Änderungen am Code hinter den Objekten vorgenommen werden können. JOGL hingegen besitzt alle Möglichkeiten, die auch die OpenGL API aufweist. Dabei wird der vom Nutzer erstellte Quellcode im Hintergrund in natives C umgewandelt und ausgeführt.

Im Rahmen dieser Arbeit wird das Modul für Java3D entwickelt, welches der Codegenerator standardmäßig enthält. Dabei wird nicht auf sämtliche Schritte der Softwareentwicklung eingegangen, sondern es werden in den einzelnen Kapiteln immer wieder die gleichen Themen besprochen; beginnend mit der Analyse der einzelnen Probleme, deren potentiellen Lösungen und der endgültigen Implementierung, welche jeweils am Codebeispiel erläutert wird. Diese Gliederung soll es dem Leser vereinfachen, die wichtigsten Probleme bei der Erstellung der Anwendung zu verstehen und führt wie ein roter Faden durch diese Arbeit.

1.2. Grundlagen

In den folgenden Abschnitten werden für das Verständnis dieser Arbeit grundlegende Begriffe und Sachverhalte vorgestellt.

1.2.1. Java3D

Bei Java3D handelt es sich um eine Erweiterung für die Java-Laufzeitumgebung. Mit ihr können Programme entwickelt werden, die eine grafische Darstellung von 3D-Objekten ermöglichen. Die Möglichkeiten reichen vom Aufbau und Darstellung von 3D-Objekten und der Umgebung bis zur Modellierung von Benutzeraktionen. Sie ist für viele Plattformen verfügbar, für die eine JRE vorhanden ist. Daneben bietet Java3D Loader an, mit denen Dateien aus anderen Programmen geladen werden können. Die Darstellung geschieht als Szenegraph, in dem keine Zyklen enthalten sein dürfen.

Die erste Version erschien im Dezember 1998 und wurde von Sun entwickelt. Nachdem die Entwicklung zwischenzeitlich eingestellt wurde, ist die Bibliothek seit 2004 als Open Source verfügbar und wird stetig erweitert, aktuell ist Version 1.5.2.

1.2.2. Szenegraph

Ein Szenegraph ist eine Erweiterung der klassischen Datenstruktur des Baumes. Er besteht wie dieser aus Knoten, die sich von der Wurzel aus immer weiter verzweigen. Besitzt ein Knoten weitere Kinderknoten, so spricht man von einem Elternknoten, sind keine Kinder vorhanden, so handelt es sich um einen Blattknoten, welcher das Ende des jeweiligen Zweiges darstellt.

Der Szenegraph erweitert diese Datenstruktur um Knotenkomponenten. Diese sind weder Eltern- noch Kinderknoten, sondern enthalten nur Daten, die für die mit ihr verbundene Komponente wichtig sind.

Jeder Knoten oder jede Knotenkomponente ist eine Instanz einer Java-Klasse und lässt sich in einer der folgenden Gruppen einordnen: Gruppenknoten (group), Blattknoten (leaf) oder Knotenkomponente (node component). Die Gruppenknoten sind

Container für weitere Knoten. Sie dienen zur Gruppierung und Strukturierung und bestimmen in einigen Fällen, wie ihr Inhalt zu interpretieren ist. Blattknoten enthalten Informationen zur Darstellung der Szene, dies beinhaltet Angaben zu den verschiedenen geometrischen Figuren, der Ausgabe von Audio, der Darstellung von Licht und der Steuerung. Knotenkomponenten enthalten Daten, die von Gruppen- und Blattknoten benötigt werden. Sie bilden selbst keine allein darstellbare Komponente.

Um die einzelnen Elemente miteinander zu verbinden, gibt es zwei Arten von Relationen; die Verbindung zwischen einem Eltern- und einem Kindsknoten (Vater-Kind-Beziehung) und die Referenz. Letztere kann von einem Gruppen-, Blattknoten oder einer Knotenkomponente ausgehen. Je nach Referenzierungsrichtung wird der Pfeil unterschiedlich ausgerichtet. Um beide Relationen zu unterscheiden, wird eine Vater-Kind-Beziehung durch einen durchgezogenen Pfeil gekennzeichnet, eine Referenz hingegen durch einen unterbrochenen Pfeil. Elemente können mehrere Referenzen besitzen, sowohl als Start- als auch als Endelement. Hingegen kann ein Gruppen- oder Blattknoten lediglich einen Vater besitzen.

Knoten, Knotenkomponenten und Relationen werden als Komponenten des Szenegraphen bezeichnet. Ab jetzt werden Knoten und Knotenkomponenten allgemein als Elemente bezeichnet.

Ein Szenegraph ist in zwei Teilbäume zu unterteilen; dem Content Branch, und dem ViewBranch. Der Content Branch enthält nur Komponenten, die für den Inhalt der Szene verantwortlich sind. Der View Branch hingegen steuert die Sicht auf den Inhalt. Bei der Modellierung im Programm spielt nur der Content Branch eine Rolle. Da beide Teilbäume immer mit einer BranchGroup beginnen müssen, ist die Wurzel des im Programm erstellbaren Szenegraphen deshalb immer eine BranchGroup und kann vom Benutzer nicht gelöscht werden.

Die folgende Abbildung zeigt einen einfachen Szenegraphen, der den grundlegenden Aufbau sowie die verschiedenen Elemente und deren Darstellungsformen enthält.

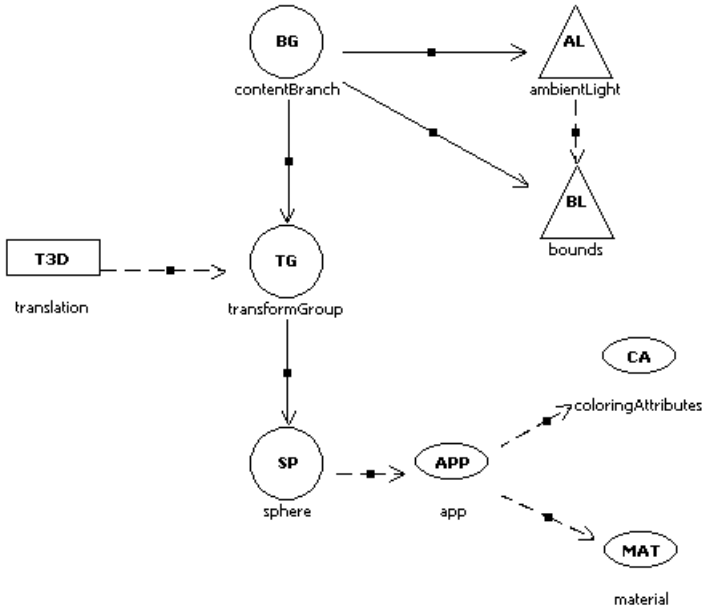


Abbildung 1.1.: Allgemeiner Szenegraph

1.2.3. Erzeugung eines Java3D-Programms

Folgende Schritte sind zur Erzeugung eines lauffähigen Java3D-Programmes nötig:

- Erstellen des Szenegraphen
- Erstellung des Quellcodes
- Übersetzen des Quellcodes in Java-Byte-Code
- Ausführen des erstellten Programms

Die letzten drei Schritte geschehen ohne Zutun des Benutzers vollautomatisch. Beim Erstellen des Szenegraphen muss der Benutzer grundlegendes Wissen über dessen Aufbau, sowie die Fähigkeiten der von ihm genutzten Komponenten aufweisen. Da das Programm nur zulässige Verbindungen erlauben soll, wird dem Benutzer bereits etwas Arbeit abgenommen. Weiterhin soll er auch darauf hingewiesen werden, wenn eine Verbindung fehlt, da ansonsten eine fehlerfreie Funktion des Programms meist nicht gewährleistet ist.

Anschließend muss der Szenegraph in Quelltext umgesetzt werden, wobei möglichst viele Kommentare dem Benutzer einen guten nachträglichen Überblick verschaffen,

wenn er den erzeugten Quelltext an eigene Bedürfnisse anpassen will.

Die erzeugten Quelltextdateien befinden sich im gleichen Ordner, in dem der Benutzer den Szenegraph gespeichert hat. Sollte er dies noch nicht getan haben, so werden die Dateien in seinem HOME-Verzeichnis abgelegt.

2. Problem-Analyse

In den folgenden Abschnitten werden die sich stellenden Probleme vorgestellt und ihre Problematik erläutert.

2.1. Daten

Daten sind die Grundlage eines jeden Szenegraphen. Sie sind nötig, um ihn zu modellieren und um Quelltext zu erzeugen. Außerdem werden sie gebraucht, um sein "Aussehen" zu beeinflussen, also die Ansicht, die der Benutzer sieht, wenn er ihn erstellt.

Um den Szenegraph speichern zu können, muss festgelegt werden, welche Daten gespeichert werden und welche neu erzeugt werden können. Dazu ist es nötig, einige Vorüberlegungen anzustellen.

Ein Szenegraph besteht wie oben erläutert aus Elementen (Knoten und Knotenkomponenten) und Kanten. In den Elementen werden Daten abgespeichert, die zur Erzeugung von Quellcode unerlässlich sind. Das sind im Allgemeinen Attribute, die zum Erzeugen einer neuen Instanz der Klasse des Elements nötig sind, aber auch die ID des Datenobjektes, welche zu dessen späterer Zuordnung nötig ist. Diese Informationen müssen gespeichert werden. Bei den Kanten ist es etwas anders. Bestünde nur der Anspruch, Quellcode zu erzeugen, müssten die Pfeile nicht gespeichert werden, sie würden kein Datenobjekt benötigen, da der Beginn und das Ende eines Pfeiles ja in den Daten der Elemente vermerkt werden.

Würde man allerdings diesen Lösungsansatz wählen, so wäre der Szenegraph für den Benutzer nicht mehr zu verstehen oder gar zu bedienen, denn es fehlen hier sämtliche

Informationen, die es dem Benutzer ermöglichen, den Szenegraph so zu strukturieren, dass er einen optimalen Überblick erhält. Dazu zählen die Möglichkeiten, Eckpunkte für Pfeile selbst zu setzen oder die Position von Elementen auf der Zeichenfläche zu bestimmen. Auch die Wahl von Zeichenfarben fällt darunter. Demzufolge müssen auch Daten, welche nur die Darstellung der Oberfläche betreffen, einbezogen werden. Bei Elementen sind dies insbesondere seine Position auf der Zeichenfläche und dessen Höhe und Breite.

Komplizierter verhält es sich bei den Pfeilen. Diese müssen ein eigenes Datenobjekt erhalten, welches neben den IDs der verbundenen Elemente auch deren Mittelpunkt, also dem Start- und Endpunkt des Pfeiles, auch noch mögliche Zwischen- oder Eckpunkte enthält. Hinzu kommen noch Angaben wie Farbe und die Angabe, ob die Eckpunkte manuell verschoben wurden.

2.2. Datenspeicherung

Die Datenobjekte müssen in geeigneter Form gespeichert werden. Da zu jeder Zeit neue Objekte hinzugefügt werden können, muss die Datenstruktur eine einfache Möglichkeit bieten, zu jeder Zeit beliebig viele neue Datenobjekte hinzufügen zu können. Dafür bietet sich eine Liste an.

Da ein Szenegraph einem Baum sehr ähnlich ist, kann die Speicherung der Daten auf den ersten Blick auf rekursivem Weg erfolgen. Bei diesem enthält jeder Knoten eine Liste mit seinen Kindsknoten. Zusätzlich würde zur Speicherung von Referenzobjekten eine weitere Liste benötigt. Nun ergibt sich aus der Definition einer Referenz, dass diese von mehreren Elementen referenziert werden kann. Demnach kann die Speicherung von Referenzen nicht auf diesem Weg erfolgen. Bei genauerer Betrachtung des Sachverhaltes kommt diese auch zur Speicherung von Knoten nicht in Frage, denn jede Operation wie das Verschieben oder Neuanlegen von Relationen würde ein Umsortieren der Daten notwendig machen. Als Beispiel kann hier das Neuanlegen eines Elements genannt werden. Dabei wird dieses per Mausklick auf der Oberfläche abgelegt. Zu diesem Zeitpunkt steht es noch mit keinem anderen Element in Verbindung. Die Speicherung muss also in einer extra dafür geschaffenen Liste erfolgen. Verbindet der Benutzer nun das Ele-

ment auf der Zeichenfläche mit einem anderen, so muss dessen Datenobjekt verschoben werden. Wird die Relation wieder gelöscht, muss das Datenobjekt wieder in die extra Liste verschoben werden, falls keine weiteren Relationen existieren. Ist das der Fall, muss es in die Liste des anderen Datenobjekts verschoben werden.

Da diese Operationen mit viel Aufwand verbunden sind, ist es sinnvoller, alle Elemente in linearer Weise zu speichern, wobei wieder getrennte Listen für Vater-Kind-Beziehungen und Referenzen verwendet werden. Vorteil dabei ist, dass die Daten in geordneter Reihe vorliegen, von der kleinsten ID bis zur größten. Das Suchen in dem Datenbestand wird dadurch vereinfacht. Außerdem können die Daten dadurch einfach in einer Datei gespeichert werden.

Nun enthält ein Element zudem noch die Angabe, mit welchen anderen Elementen es in Relation steht, da diese Informationen mehrfach benötigt werden. Dazu bieten sich ebenfalls aus obigen genannten Grund Listen an, die hier allerdings nur die IDs der jeweiligen Elemente beinhalten müssen, jeweils eine für die Kindsknoten und zwei für die Referenzen. Letzterer Sachverhalt lässt sich damit begründen, dass eine Relation in zwei Richtungen existieren kann. Bei einer Vater-Kind-Beziehung bindet der Elternden Kindsknoten ein, hierbei ist nur eine Richtung existent. Eine Referenz hingegen kann von einem Element ausgehen oder an diesem enden. Bestimmte Knotenelemente sind kein Endknoten einer Referenz, und würden somit beim rekursiven Erzeugen von Quellcode nicht erfasst werden. Demnach muss also auch der Endknoten einer bei ihm endenden Referenz deren Startknoten kennen, damit dieser erzeugt werden kann.

Um die Daten permanent zu speichern, ist ein Verfahren notwendig, mit dem die Java-Objekte in einer Datei gespeichert werden können. Dazu bieten sich zwei Verfahren an: die Speicherung in Form von serialisierten Objekten und als XML.

Deren Eigenschaften werden nun in folgender Tabelle einmal gegenübergestellt.

	Serialisierte Objekte	**XML**
Lesbarkeit	binär, nicht lesbar	gut lesbar
Kompatibilität zu neuen Programmversionen	schlecht, bei Änderungen an der Struktur der Datenobjekte können alte Dateien nicht mehr geladen werden	einfache Anpassung von bestehenden Dateien möglich
Implementierung	einfach, Datenobjekte und Attribute müssen lediglich Interface Serializable implementieren	kompliziert, Datenobjekte müssen individuelle Methoden für Speichern und Laden von XML bereitstellen

Tabelle 2.1.: Vergleich von Speicherformaten

Aufgrund der besseren Lesbarkeit und der Kompatibilität zu neuen Programmversionen ist XML als Dateiformat zu favorisieren. Da das Hauptaugenmerk der Entwicklung allerdings zuerst auf der Erzeugung von Quellcode liegen soll, wird zunächst die Speicherung in Form von serialisierten Objekten implementiert. Falls es die Entwicklungszeit zulässt, wird auch die Speicherung in XML realisiert.

2.3. Referenzen

Eine Referenz stellt eine Relation zwischen Elementen eines Szenegraphen dar, die keine Vater-Kind-Beziehung ist. Sie wird verwendet um zu signalisieren, dass ein Element von dem anderen benötigt wird, ohne dabei dessen Kind zu sein.

Dabei gibt es zwei Arten von Referenzen, erforderliche und optionale. Der Unterschied zwischen beiden ist, dass optionale Referenzen einfach weggelassen werden können. Für diese wird dann ein Standardparameter verwendet, wobei dieser Parameter oft auch auf NULL gesetzt wird. Eine optionale Referenz wird als Ellipse dargestellt.

Wird zum Beispiel einer Appearance-Knotenkomponente keine Material-Knotenkom-

ponente übergeben, so werden alle Beleuchtungsoptionen für diese Appearance deaktiviert. Dabei ist das Programm aber kompilierbar. Diese Vorgehensweise ist auch in der Hinsicht sinnvoll, dass es allein für eine Appearance elf ergänzende Knotenkomponenten gibt. Müssten diese alle beim Erstellen angegeben werden, würde dies schnell zu Unübersichtlichkeit führen.

Andererseits gibt es Referenzen, welche zwingend benötigt werden und daher nicht weggelassen werden dürfen, denn ohne sie wäre die Funktion des referenzierten Elements nicht gewährleistet. Als Beispiel ist hier der RotationInterpolator anzuführen. Er benötigt eine Alpha- und eine Transform3D-Knotenkomponente, welche auch mit Standardparametern initialisiert sein können. Diese erforderlichen Elemente werden dann mit einem Rechteck dargestellt.

Allerdings ist auch in offiziellen Quellen die Darstellung nicht durchgängig. Damit der Benutzer trotzdem unterscheiden kann, von welcher Art die Referenz ist, wird im Programm die oben genannte Trennung verwendet. Einen Unterschied im Quelltext gibt es nicht, da eine Einbindung einer Referenz immer durch den Startknoten geschieht.

2.4. Semantik eines Szenegraphen

Mit den zwei Relationen Vater-Kind-Beziehung und Referenz existieren zwei Möglichkeiten, Elemente miteinander zu verbinden. Eine solche Verbindung stellt sich im Quelltext meistens durch den Aufruf einer Methode mit einem der Elemente als Parameter dar. Jedoch besitzen die verschiedenen Klassen des Java3D-Paketes nicht alle die gleichen Methoden, sondern nur solche, die für sie sinnvoll sind.

Im Umkehrschluss bedeutet dies für den Szenegraphen, dass nicht alle Elemente miteinander verbunden werden dürfen. Es müssen Regeln gefunden werden, welche das gültige Verbinden von Knoten und Knotenkomponenten durch Vater-Kind-Beziehungen und Referenzen beschreiben. Dabei kann man sich anfangs an den Namen der jeweiligen Knoten und Beziehungen orientieren.

Dadurch können schnell offensichtliche Regeln gefunden werden, die sich bereits aus der Bezeichnung von Knoten herleiten; zum Beispiel die Tatsache, dass ein Blattknoten keine Kindsknoten haben oder ein Gruppenknoten mehrere Kinder besitzen kann.

Daneben gibt es Regeln, die sich weniger schnell erschließen. Anders als die oben genannte Vater-Kind-Beziehung kann eine Referenz per Definition zwischen beliebigen Elementen gezogen werden, wobei jedes Element mehrere Referenzen aufweisen kann. Dies mag zwar syntaktisch korrekt sein, semantisch hingegen ergeben nicht alle einen Sinn.

Auch bei einer Vater-Kind-Beziehung gibt es Ausnahmen, in denen keine Verbindung zwischen zwei Knoten möglich ist. Demzufolge muss ein Weg gefunden werden, wie bereits bei der Erstellung eine irreguläre Verbindung verhindert werden kann. Die Umsetzung muss dabei so erfolgen, dass der Benutzer bereits beim Anlegen einer Relation darauf hingewiesen wird, ob diese erstellt werden kann oder nicht.

Anders als im Fall von fehlenden sind doppelt vorhandene Referenzen meist nicht problematisch. Die zuletzt hinzugefügte Referenz überschreibt im Quelltext die zuvor angelegte. Nur in bestimmten Ausnahmefällen sind mehrere Referenzen auf verschiedene Objekte eines Typs möglich und erwünscht. Deshalb muss das Programm kontrollieren, wie viele Referenzen bereits von einem Element auf einen Typ erstellt wurden. Ist die maximale Anzahl erreicht, darf keine weitere angelegt werden.

2.5. Oberfläche des Codegenerators

Die Oberfläche des Programms muss so aufgebaut werden, dass sie sich dem Benutzer intuitiv erschließt. Dazu zählen neben Menüs mit Standardaufgaben auch Toolbarelemente, die durch bekannte Symbole wie ”Speichern“ auf sich aufmerksam machen.

Das Hauptaugenmerk liegt aber auf der ”Zeichenfläche“, also dem Bereich, wo der Szenegraph modelliert, ”gezeichnet“ wird. Er wird im Weiteren als Zeichenfläche angegeben. Hier werden die verschiedenen Elemente des Szenegraphen vom Benutzer angelegt und bearbeitet. Dazu zählen Knoten, Knotenkomponenten und Kanten beziehungsweise Pfeile. Alle besitzen Eigenschaften, die ihr Aussehen in der Oberfläche steuern und die der Benutzer nach Belieben beeinflussen kann.

Zunächst sollten die Elemente je nach ihrem Typ unterschiedlich dargestellt werden. Ein Gruppenknoten wird durch einen Kreis, ein Blattknoten durch ein Dreieck und

eine Knotenkomponente entweder durch ein Rechteck oder eine Ellipse dargestellt. Weiterhin sollten der Name und der Typ des Knotens oder der Knotenkomponente angezeigt werden. Sie sollten in ihrer Größe änderbar und in ihrer Position verschiebbar sein.

Neben diesen, nur für die Oberflächendarstellung relevanten Daten, muss für jedes Element eine Eingabemöglichkeit für Attribute vorhanden sein, die zum Erzeugen der betreffenden Klasse nötig sind. Dazu bietet sich ein Fenster an, dass per Rechtsklick auf das Element geöffnet wird. Dort können die Attribute eingegeben werden. Weiterhin gibt es die Möglichkeit, die für die Oberfläche relevanten Daten, also den Namen des Knotens und seine Darstellungsfarbe, zu ändern. Der Name steht hierbei sinnvollerweise auch für die Bezeichnung des Attributs im Quelltext.

Attribute gleichen Namens dürfen nicht doppelt vorkommen, da der Quelltext für den Szenegraphen mit einer einzigen Methode erzeugt wird. Es muss demnach sichergestellt werden, dass in einem Szenegraph keine Knoten mit doppelten Namen angelegt werden können.

Kanten (oder Pfeile) werden ähnlich behandelt. Standardmäßig wird ein Pfeil gerade zwischen End- und Startpunkt gezeichnet. Wenn aber viele Kanten gezeichnet werden, wird die Darstellung dadurch schnell unübersichtlich. Es ist besser, wenn Pfeile möglichst geordnet zwischen den Komponenten verlaufen, wobei jeder Benutzer eine unterschiedliche Vorstellung von Ordnung hat. Dazu müssen beliebig viele Zwischenpunkte erstellt werden können. Ein Zwischenpunkt ist ein Punkt auf dem Pfeil, der zwischen Start- und Endpunkt liegt. Jeder dieser Punkte markiert eine Ecke des Pfeiles. Diese können beliebig auf der Zeichenfläche verschoben werden. Im Eigenschaftsfenster eines Pfeiles muss also eine Möglichkeit vorhanden sein, die Anzahl der Zwischenpunkte zu verändern. Will der Benutzer zur ursprünglichen Darstellung zurückkehren, reduziert er die Anzahl der Zwischenpunkte wieder auf das Minimum von Eins. Verschiebt er nun aber ein Element, bleibt der Zwischenpunkt an genau dieser Position. Damit der Pfeil wieder in einer Linie gezeichnet werden kann, muss im Datenobjekt des Pfeiles vermerkt werden, dass die automatische Routenführung wieder genutzt wird. Dazu bietet sich ein Wahrheitswert an, da die Angabe nur zwei Zustände kennt, ”Autoroute“ und manuell.

Die Farbe eines Pfeiles kann wie die eines Knotens ebenfalls geändert werden. Je nach Art der Kante wird sie unterschiedlich dargestellt. So wird eine Vater-Kind-Beziehung durch einen durchgezogenen Pfeil symbolisiert, eine Referenz durch einen gestrichelten Pfeil.

Will der Benutzer mehrere Elemente verschieben, so muss sowohl das Verschieben eines Elements, als auch von mehreren Elementen möglich sein. Als Auswahlmöglichkeit kommen sowohl die Wahl per Maus als auch per Hotkey in Frage.

Da das Programm modular aufgebaut werden soll, bietet sich das Hinzufügen von Elementen über ein Menü in der Menüleiste an, denn so kann jedes Modul sein eigenes Menü mitbringen. Beim Laden des Programms wird dieses in die Menüzeile eingebunden. Sinnvoll ist es, wenn sich der Aufbau an der Art der Knoten und an deren Bedeutung orientiert. Zusätzlich zu den Namen können Symbole auf die Zugehörigkeit des betreffenden Knotens oder der Knotenkomponente hinweisen. Es wäre auch möglich, die Klassen nach ihrem Paket einzuordnen. In der aktuellen Java3D-Version sind jedoch die meisten Klassen ungeordnet in einem Paket untergebracht. Diese alle in einem Untermenü unterzubringen würde die Übersichtlichkeit extrem einschränken.

Beim Erstellen eines Szenegraphen wird der Benutzer Elemente oft mehrfach benötigen. Beispiele dafür sind die Branch- und TransformGroup. Jedes Element einzeln aus dem Menü einzufügen, ist nicht praktikabel, da dafür viele Klicks im Menü erforderlich sind. Das Programm sollte eine Möglichkeit bereitstellen, häufig benutzte Elemente schneller einzufügen. Dafür bietet es sich an, eine weitere Toolbar unter der Menüleiste anzuordnen, welche jeweils die zuletzt verwendeten Elemente als Schnellzugriff bereitstellt. Da in dieser Leiste jedoch nicht unbegrenzt Platz sein wird, müssen die Buttons eine Angabe besitzen, wie oft sie seit ihrem Anlegen benutzt wurden. Damit werden nicht nur die zuletzt verwendeten Elemente angezeigt, sondern die mit der höchsten Zugriffszahl.

Das Programm soll den Benutzer des Weiteren dabei unterstützen, fehlerhafte Eingaben einfach zu entdecken und zu korrigieren. Dabei muss gewährleistet werden, dass der Benutzer nicht noch einmal alle Eingaben tätigen, sondern lediglich den fehlerhaften Wert ändern muss.

2.6. Generierung von Quellcode

Um ein funktionsfähiges Java-Programm zu erzeugen und es starten zu können, benötigt der Benutzer neben kompilierbaren Quellcode folgende Softwarepakete, jeweils möglichst in aktueller Version:

- eine auf dem System installierte JRE aus der 1.6er-Reihe
- ein JDK aus der 1.6er-Reihe
- die Java3D-Runtime-Erweiterungaus der 1.5er-Reihe

Eine Ausnahme bildet dabei das Java3D-Softwarepaket. Aufgrund dessen, dass der Codegenerator nur mit dem aktuellen Softwarestand entwickelt wird, ist ein Update der Java3D-Version nur mit Einschränkungen möglich.

Ein Fehlen der Pakete macht sich unterschiedlich bemerkbar. Während ohne eine installierte JRE der Codegenerator nicht startet, wird das Fehlen der anderen Pakete erst beim Kompilieren des Quellcodes vom Codegenerator entdeckt. In diesem Fall wird der Benutzer vom System durch eine Warnmeldung auf diesen Umstand hingewiesen.

Sind alle Voraussetzungen erfüllt, kann der Quellcode erstellt und kompiliert werden.

Jedes Element des Szenegraphen ist eine Instanz einer Java3D-Klasse. Um es zu konstruieren, ist im einfachen Fall kein weiteres Zutun seitens des Benutzers erforderlich. Einige Klassen benötigen keine zusätzlichen Attribute zur Erstellung, zum Beispiel die BranchGroup. Es gibt jedoch auch Klassen, für welche es zwingend weiterer Angaben bedarf, damit diese ihre Aufgaben korrekt erfüllen können. Dabei existieren zwei Arten von Attributen, Elementar- und Referenztypen.

Elementartypen sind solche, die nicht mit Hilfe des new-Konstrukts erstellt werden müssen, wie z.B. int. Viele der 3D-Klassen besitzen solche und jedes dieser Attribute wird anfangs mit seinem Standardwert initialisiert. Dieser wird zur Erzeugung verwendet, wenn der Benutzer keine Änderung vornimmt. Da diese Werte oft schon sinnvoll gewählt sind und keiner Änderung bedürfen, ergibt sich daraus bereits eine Vereinfachung für den Benutzer. Er muss lediglich die von ihm gewünschten Werte modifizieren.

Wird zur Erstellung ein Referenzobjekt benötigt, wie ein anderes Java3D-Objekt, muss dieses existieren. Es ist deshalb notwendig, vor der Erstellung zu prüfen, ob al-

le diese Objekte, im Szenegraph Knoten oder Knotenkomponenten, vorhanden sind. Ist dies nicht der Fall, muss der Benutzer entsprechend darauf hingewiesen werden. Daneben gibt es noch weitere Besonderheiten, die beachtet werden müssen; wobei die sogenannten Capabilities eine besondere Stellung einnehmen. Diese definieren, inwiefern auf bestimmte Eigenschaften Zugriff genommen werden kann, wenn der zugehörige Knoten bereits kompiliert wurde und in der Szene dargestellt wird. Bestimmte Knoten oder Knotenkomponenten, die das Aussehen oder die Geometrie beeinflussen, benötigten ein gesetztes Bit, um korrekt zu funktionieren.

Damit der Quellcode vom Compiler übersetzt werden kann, muss er der Java-Syntax entsprechen. Dabei ist es wichtig, dass der Benutzer den Quellcode leicht verstehen kann. Dieser muss demnach übersichtlich strukturiert sein. Dazu gehört, dass die rekursive Vorgehensweise sich in einer Einrückung von Zeilen darstellt. Der Szenegraph stellt sich also auch im Quellcode als Baumstruktur dar. Um das Verständnis weiter zu erhöhen, werden vor jedem wichtigen Abschnitt Kommentare in den Quelltext eingefügt.

Die Erzeugung von Quellcode beginnt immer bei der Wurzel. Aufgrund dessen, dass der ContentBranch immer aus einer Instanz der Klasse BranchGroup bestehen muss, ist eine solche auch immer die Wurzel des Szenegraphen. Sie darf nicht gelöscht werden, da dem Programm sonst der Einstiegspunkt fehlt. Es muss also vom Programm dafür gesorgt werden, dass der Nutzer diese nicht versehentlich löscht. Ansonsten allerdings kann sie wie jedes andere Element bearbeitet werden.

Anschließend wird rekursiv vorgegangen, wobei beim Erzeugen der Referenzen beachtet werden muss, dass benötigte Referenzobjekte auch zum Zeitpunkt der Erzeugung zur Verfügung stehen. Falls ein Knoten oder eine Knotenkomponente bereits erzeugt wurde, dürfen diese im Falle einer späteren weiteren Referenzierung nicht noch einmal angelegt werden. Demnach muss ein Hinweis enthalten sein, dass der Code bereits generiert wurde, da ansonsten eine doppelte Deklarierung des Attributes möglich ist, was zu einer Beanstandung durch den Compiler führt.

Die Java3D-Umgebung liegt aktuell in Version 1.5.2 vor, mit welcher der Codegenerator entwickelt wird. Das Modul ist prinzipiell also nur mit dieser Version kompatibel. Wird es mit einer neuen Version verwendet, kann es sein, dass der generierte Quellcode

nicht mehr kompilierbar ist. Dies geschieht allerdings nur, wenn verwendete Methoden und Klassen grundlegend geändert wurden, was zumeist nicht der Fall ist. Bei größeren Versionssprüngen muss das Modul aber erneut auf Kompatibilität getestet werden.

Bei der Entwicklung ist es notwendig, mit Hilfe der Java3D-API-Dokumentation für jede Java3D-Klasse eine Klasse für den Codegenerator zu erstellen, die die benötigten Attribute mit ihren Standardwerten enthält. Außerdem muss sie eine Methode besitzen, die den Quellcode für diese 3D-Klasse erzeugt.

Dabei werden jedoch nicht alle Klassen der API berücksichtigt, da nicht jede als Knoten oder Knotenkomponente in einem Szenegraphen vorkommen kann. Einige sind abstrakt und vererben Funktionen, demzufolge werden nur die erbenden Klassen verwendet. Andere Klassen sind lediglich Hilfsmittel und werden nur im Quellcode genutzt, zum Beispiel die Klassen zur Vektorrechnung.

Klassen und Funktionen, die als Deprecated, also veraltet eingestuft sind, werden vom Programm nicht unterstützt. Für diese gibt es anderweitigen Ersatz und sie können mit jeder weiteren Version des Java3D-Pakets entfallen. Aufgrund dessen, dass der Codegenerator eine Neuentwicklung ist, wird diese Unterstützung nicht benötigt.

3. Lösungskonzeption / Programm-Entwurf

3.1. Lösungsansatz

3.1.1. Daten

Um die Daten eines Szenegraphen zu verwalten, wird für jeden Datentyp eine eigene Klasse erstellt; also eine für ein Element und eine für die Relationen oder Pfeile. Zunächst soll die Klasse für einen Pfeil näher erläutert werden.

Datenobjekt eines Pfeils Ein Pfeil wird von einem Element zu einem anderen angelegt. Dabei ist es egal, ob es eine Referenz oder eine Vater-Kind-Beziehung ist.

Um seine Aufgabe korrekt erfüllen zu können, benötigt er die in folgenden Abschnitten beschriebenen Attribute:

Typ Das Aussehen wird durch den Typ geregelt, der entweder eine Referenz oder eine Vater-Kind-Beziehung sein kann.

IDs seines Start- und Endelements sowie deren Mittelpunkte Daneben hinaus muss der Pfeil Angaben über die IDs seines Start- und Endelement besitzen. Die Begründung liegt darin , dass wenn eine von diesen verschoben, vergrößert oder gelöscht wird, so müssen die mit ihnen verbunden Pfeile identifiziert und je nach durchgeführter

Aktion aktualisiert werden. Das bedeutet im Einzelnen, dass bei einer Verschiebung oder einer Änderung der Größe der Start- oder der Endpunkt des Pfeiles aktualisiert wird. Deshalb muss neben der ID auch diese Angabe enthalten sein.

Ein Pfeil wird immer vom Mittelpunkt des Start- zum Mittelpunkt des Endelements gezeichnet. In der Darstellung verschwindet dieser unter dem Element selbst. Am Schnittpunkt mit dem Endendelement wird eine Pfeilspitze gezeichnet, um die Richtung des Pfeiles anzuzeigen.

Zwischenpunkte in absoluter und relativer Form Damit auch Pfeile mit mehreren Zwischenabschnitten gezeichnet werden können, muss jeder Pfeil zwei Listen mit Zwischen- oder Eckpunkten besitzen, die diese sowohl in absoluter wie auch in relativer Form enthalten. Absolute Punkte beziehen sich auf das Koordinatensystem des Parent-Containers; dass heißt im Fall der Pfeile auf die Zeichenfläche. Mit ihnen wird dann der Pfeil in Etappen gezeichnet; zuerst vom Startpunkt zum ersten Zwischenpunkt, danach zwischen den einzelnen weiteren Zwischenpunkten und zuletzt vom letzten Zwischen- zum Endpunkt. Diese Angabe allein reicht jedoch nicht aus: Wird ein Pfeil mit mehreren Elementen markiert und verschoben, so dürfen die relativen Abstände zueinander nicht verändert werden. Nur mit den absoluten Angaben ist dies nicht möglich, weshalb eine weitere Liste mit den relativen Abständen zum Startelement vorhanden sein muss. Wird das Startelement verschoben, werden diese Abstände und die absoluten Punkte aktualisiert.

Anfangs wird ein Pfeil nur mit einem Zwischenpunkt initialisiert. Der Benutzer kann nach Belieben Punkte hinzufügen oder löschen, welche jeweils in der Mitte der größten Zwischenstrecke angelegt werden. Gelöscht wird immer die kleinste Zwischenstrecke.

Angabe zur Nutzung der automatischen Routenführung Standardmäßig ist die sogenannte "Autoroute"-Funktion aktiv, welche die Zwischenpunkte immer auf einer Gerade zwischen Start- und Endpunkt anordnet. Verschiebt der Benutzer einen dieser Punkte, so wird die Funktion abgeschaltet. Er kann sie jedoch im Eigenschaftsdialog des Pfeiles wieder aktivieren, wodurch der Pfeil wieder auf eine Linie zwischen beiden Punkten gesetzt wird. Die Anzahl der Zwischenpunkte bleibt allerdings konstant.

Farbe des Pfeils Der Eigenschaftsdialog muss außerdem eine Möglichkeit bieten, die Farbe eines Pfeiles zu ändern. Auch diese Angabe wird in den Daten vermerkt. Standardmäßig ist die Farbe eines jeden Pfeiles Schwarz.

Datenobjekt eines Knotens oder einer Knotenkomponente Ein Element ist eine Instanz einer Java3D-Klasse. Alle besitzen grundsätzlich die gleichen Eigenschaften, unterscheiden sich lediglich in den Attributen und dem zu erzeugenden Quelltext. Es bietet sich also an, dass alle Datenobjekte der Elemente von einer zentralen Klasse erben. Sie stellt zentrale und für alle Elemente gültige Methoden und Attribute zur Verfügung. Das Datenobjekt eines Elements muss dann lediglich einen Konstruktor zur Erstellung und aufgabenspezifische Methoden bieten. Im Allgemeinen ist dies nur die Methode zum Erstellen von Quellcode.

Weiterhin können die Datenobjekte individuelle Angaben enthalten. Als Beispiel sei hier die Klasse zur Darstellung eines BoundingLeaf zu nennen. Sie stellt einen Blattknoten dar, der Informationen über die Grenzen einer Region enthält. Er wird benötigt, um für bestimmte Knoten einen Gültigkeitsbereich festzulegen, für die dessen spezielle Eigenschaften gelten. Nun kann eine Region in Java3D von unterschiedlichem Typ sein. Als häufigste Form wird die BoundingSphere verwendet, welche eine Kugel mit einem Mittelpunkt und einem Radius darstellt. Alle Objekte, die innerhalb des Radius liegen und die mit dem Knoten mit dem Einflussbereich verbunden sind, werden von dessen Eigenschaften beeinflusst. Weitere Typen von Regionen sind die BoundingBox, also eine Region in Form eines Rechtecks und ein BoundingPolytope. Die Angabe, welcher Typ benutzt werden soll, wird nur von dieser Klasse benötigt und auch nur von ihr gespeichert.

Benötigt ein Datenobjekt wie der genannte BoundingLeaf zusätzliche Attribute, so muss er die Methoden zum Speichern und Laden von XML der Elternklasse überschreiben. Diese müssen dann die Standardattribute und die zusätzlichen berücksichtigen.

Je nachdem, ob das Element ein Knoten oder eine Knotenkomponente ist, werden bestimmte Attribute nicht benötigt; zum Beispiel wird eine Knotenkomponente die IDs der Kindsknoten nicht benötigen.

Um die individuellen Daten zu speichern, ist es sinnvoll, dass die vererbende Klasse

eine Methode bereitstellt, die diese Angaben speichert. Die Begründung liegt darin, dass bei der Implementierung in jedem Datenobjekt lediglich diese Methode aufgerufen werden muss und somit weniger Quelltext entsteht. Diese Angaben sind: die ID, die Farbe, der Typ des Elements, die Klassen-ID, der Anzeigename für die Oberfläche und die maximale Anzahl von Elternknoten.

In den folgenden Abschnitten werden die einzelnen Attribute kurz vorgestellt.

ID Die ID wird beim Erzeugen des Elements gesetzt und ist lebenslang gültig. Sie wird auch beim Löschen nicht wieder freigegeben.

TypID Die Typ-ID regelt, welchen Typ der Knoten oder die Knotenkomponente besitzt. Für jeden der drei Typen Gruppenknoten, Blattknoten und Knotenkomponente muss dafür eine Konstante definiert werden. Die Typ-ID wird beim Erzeugen des Knotens gesetzt und kann dann nicht mehr geändert werden.

KlassenID Die Klassen-ID ist eine Konstante, mit welcher festgestellt werden kann, welche Klasse der Knoten oder die Knotenkomponente darstellt. Sie wird unter anderem benötigt, damit sichergestellt werden kann, dass nur Elemente verbunden werden, die auch verbunden werden dürfen.

Klassenname Der Klassenname ist ein in der Regel zwei- bis dreistelliges Buchstabenkürzel, dass die Klasse des Elements dem Benutzer verdeutlicht. Dabei kann ein bestimmter Buchstabe bereits die Zugehörigkeit zu einer bestimmten Gruppe signalisieren. Das Kürzel "RI" steht für "RotationInterpolator", "PI" für "PositionInterpolator".

Name des Elements Die Bezeichnung eines Elements ist gleichzeitig auch der Name der Variable im Quelltext. Es bietet sich an, ihn anfangs nach dem Initialisieren immer nach dem Muster "element + ID" bilden zu lassen, um die Forderung der Eindeutigkeit zu gewährleisten. Er kann später vom Nutzer geändert werden. Dabei wird geprüft, ob er der Java-Namenskonvention entspricht. Falls nicht, wird er angepasst. Das Programm prüft außerdem, ob der Name bereits von einem Element verwendet wird. In diesem Fall muss der Benutzer einen neuen Namen eingeben.

Oberflächenspezifische Daten Damit die Darstellung des Elements auch nach einem Speicher- und Ladevorgang noch identisch ist, werden die folgenden Daten ebenfalls gespeichert:

- Farbe
- Position auf der Zeichenfläche
- Höhe
- Breite

Attribute des Elements Die zur Erzeugung benötigten Attribute müssen ebenfalls gespeichert werden. Diese werden bereits beim Anlegen des Objektes generiert und werden zur Laufzeit nicht mehr geändert. Auch sie werden in einer Liste gespeichert.

Dieses Liste enthält keine Attribute, welche selbst ein Element des Szenegraphen sind. Sie werden per Referenz mit dem Element verbunden.

Liste mit IDs der Kindsknoten Eine weitere Liste enthält die Kinder der Komponente, insofern sie Kinder besitzen kann, also ein Gruppenknoten ist. Die Kinder sind dabei in Form ihrer ID hinterlegt, da diese sich einfach abspeichern lässt.

Listen mit den IDs der Referenzobjekte Ähnlich verhält es mit den IDs für die Elemente, mit denen das aktuelle Element durch eine Referenz verbunden ist. Dabei werden zwei Listen benötigt; eine für die Referenzobjekte, die Endelement einer Referenz sind und eine weitere für Referenzobjekte, welche Startelement einer Referenz sind. Zur Erzeugung von Quellcode ist es notwendig, dass ein Element auch Elemente erkennt, bei welchen es das Endelement der jeweiligen Referenz ist.

Liste mit den Elternknoten Eine weitere Liste speichert die IDs der Elternknoten. Obwohl im Allgemeinen nur ein Elternknoten vorhanden ist, wurde für eventuelle spätere Erweiterungen hier eine Liste zur Speicherung genutzt.

Maximale Anzahl der Elternknoten Die maximale Zahl von Elternknoten muss erfasst werden, damit beim Anlegen einer Vater-Kind-Beziehung geprüft werden kann, ob der aktuell vom Benutzer angeklickte Knoten bereits einen Vater hat. Wenn dies der Fall ist, darf keine weitere Relation angelegt werden.

Diese Angabe variiert je nach Typ. Standardmäßig hat ein Gruppen- oder Blattknoten nur einen Vater, das Attribut hat also den Wert 1. Wird eine Vater-Kind-Beziehung mit jenem Knoten angelegt, wird der Zähler dekrementiert und es kann so keine weitere Relation erstellt werden. In einigen Fällen darf auch ein Gruppenknoten keinen Vater haben; das Attribut erhält den Wert 0.

Bei Knotenkomponenten ist dieser Wert immer auf -1 gesetzt.

Erlaubte Kindsknoten Einige Gruppenknoten dürfen nur bestimmte Gruppen- und Blattknoten als Kinder besitzen. Deren IDs werden in einer extra Liste gespeichert. Beim Anlegen einer Vater-Kind-Beziehung prüft das Programm, ob diese zulässig ist.

Erlaubte Referenzen Analog zu den erlaubten Kindsknoten gilt das gleiche Prinzip auch für Referenzen. Dazu werden die IDs der Elemente benötigt, die vom aktuellen Element referenziert werden dürfen. Beim Herstellen einer Referenz zwischen zwei Elementen in der Oberfläche wird diese Liste beim Hinzufügen des Endelements abgefragt. Ist sie leer, darf keine Referenz vom aktuellen Element erstellt werden. Enthält sie IDs, muss die des gewünschten Endelements enthalten sein, damit die Referenz erstellt werden kann.

Um zu kontrollieren, wie viele Referenzen von einem Element bereits zu Objekten eines bestimmten Typs erstellt worden sind, muss der Vektor neben der ID auch noch eine Angabe über die maximale Anzahl von Referenzen zu diesem Typ von Objekten enthalten. Dafür bietet sich eine eigene Klasse an, die die benötigten Angaben per Methodenaufruf bereitstellt.

Erforderliche Referenzen Damit beim Erzeugen des Quellcodes alle benötigten Referenzen vorhanden sind, müssen die unbedingt erforderlichen Referenzobjekte jedes Elements in diesem vermerkt sein. Dazu bietet sich wieder eine Liste mit den IDs der

Referenzobjekte an. Nur wenn ein Element Referenzen mit allen IDs in diesem Vektor enthält, wird der Code erstellt.

Damit der Benutzer über fehlende Referenzobjekte benachrichtigt werden kann, muss hier ein Name für die Fehlermeldung enthalten sein. Da auch hier wieder die ID erforderlich ist, kann dafür die gleiche Klasse wie für die erlaubten Referenzobjekte benutzt werden. Sie ist dazu um ein weiteres Attribut und eine Methode für den Namen der Klasse zu ergänzen.

Erforderliches oder optionales Referenzobjekt Speziell für die Knotenkomponenten regelt ein Attribut die Darstellung. Wird eine Knotenkomponente von einem Element zwingend benötigt, so wird sie mit einem Rechteck dargestellt, ansonsten mit einer Ellipse. Weitere Auswirkungen hat dies allerdings nicht, denn beim Erzeugen von Quellcode besteht zwischen beiden kein Unterschied.

Weitere Attribute Außerdem besitzt das Datenobjekt noch weitere Hilfsvariablen, die temporäre Angaben speichern; wie zum Beispiel die Angabe, ob der Quellcode bereits generiert wurde oder einen Hilfetext im Falle eines Fehlers.

Darstellung eines Attributs Je nach gewünschtem Typ des Attributs bieten sich hier Oberflächenkomponenten zur Eingabe an. Ein Attribut vom Typ Boolean lässt sich am besten mit einer Checkbox darstellen; ein String am Besten mit einem Textfeld. Komplizierter wird es, wenn die interne Darstellung eines Attributes von der äußeren, also von der für den Benutzer sichtbaren, abweicht.

Ein Beispiel dafür findet sich in der Klasse Alpha. Diese erstellt aus einem Zeitwert einen Wert zwischen Null und Eins. Damit lassen sich einfach Operationen ausführen, bei welchen ein Wert periodisch geändert werden soll. Der Modus, also ob vorwärts oder rückwärts vorgegangen wird, regelt beim Erstellen des Objekts der Parameter ”int mode“. Er kann entweder aus dem Wert ”Alpha.DECREASING_ENABLE“ für rückwärts, oder ”Alpha.INCREASING_ENABLE“ für vorwärts bestehen. Im Quelltext muss entweder die Zahl im Klartext, oder eine der genannten Bezeichner stehen. Bei der Eingabe ist keines von beiden optimal, denn mit einer Zahl kann der Benutzer in der Regel nichts anfangen. Des Weiteren sind die Bezeichner zu lang, denn auf

das ”Alpha.“ kann verzichtet werden. Ideal sind daher Bezeichnungen wie ”INCREASING_ENABLE“.

Die beste Eingabemöglichkeit für Attribute, für die nur spezielle Werte gültig sind, sind Comboboxen. Java bietet keine Möglickeit, zwischen dem dargestellten Text und dem Wert dahinter zu unterscheiden. Anders als zum Beispiel in C#, wo der Benutzer einen angezeigten Text einem dahinter stehenden Key oder Value zuordnen kann, gibt es diese Option in Java nicht. Deshalb müssen solche Attribute zusätzliche Parameter erhalten, die eben genau diese Angaben enthalten. Im Beispiel der Alpha-Modi bestehen diese aus zwei Listen. Sie enthalten interne, also mit vorangestelltem ”Alpha.“ und externe Darstellungen. Das Attribut weist dann genau die interne Bezeichnung mit dem gleichen Index wie die gewählte externe Bezeichnung auf.

Daneben muss dem Benutzer mitgeteilt werden, wie die Bezeichnung des Attributes lautet. Dies geschieht am Besten über ein, der Eingabekomponente vorangestelltes, Label.

Die Darstellung erfolgt für jedes Attribut in einem eigens dafür erstelltem Attribut-Datensatz. Dieser besteht aus einem Container, welcher das Label und die Eingabekomponente enthält. Er muss darüber hinaus Methoden bieten, um den vom Benutzer eingegebenen Wert auf Gültigkeit zu prüfen, ihn in den vorgesehenen Datentyp umzuwandeln und die Eingabekomponente beim Laden mit diesem Wert zu füllen.

Syntaxprüfung Bei bestimmten Typen von Attributen ist die freie Eingabe von Werten und somit auch die Falscheingabe möglich. Andere Typen erfordern zudem eine bestimmte Syntax bei der Eingabe; zum Beispiel die Klasse Vector3f, welche einen mathematischen Vector mit drei Float-Werten darstellt. Die Eingabe erfolgt hierbei in einem Textfeld; die einzelnen Werte werden dabei durch ein ”;“ getrennt. Damit mögliche Eingabefehler erkannt und der Benutzer darauf hingewiesen werden kann, muss die Klasse zusätzliche Methoden besitzen, die je nach Typ der Eingabekomponente und des Attributes prüft, ob der vom Benutzer angegebene Wert korrekt ist. Wird ein Fehler entdeckt, muss das Programm den Benutzer entsprechend darauf hinweisen. Dies kann am Besten geschehen, indem das Eingabefeld rot eingefärbt wird. Eine Speicherung ist erst möglich, nachdem der Fehler korrigiert wurde.

Zur Prüfung auf syntaktische Korrektheit werden reguläre Ausdrücke verwendet, die dem Attribut beim Erzeugen übergeben werden. Beim endgültigen Speichern des Eingabewertes in der dafür vorgesehen Variable wird dieser in das korrekte Format geparst. Dabei wird in obigem Beispiel des Vector3f-Objekts der String aus dem Textfeld ausgelesen und in drei Teilstrings zerlegt. Diese werden danach in einen Float-Wert umgewandelt, mit welchen anschließend ein neues Vector3f-Objekt angelegt wird.

3.1.2. Datenspeicherung

Damit die Daten auch dauerhaft verfügbar sind, müssen sie gespeichert werden. Dazu wird eine eigene Klasse genutzt, die Methoden zum Speichern und Laden bereit stellt.

Speichern der Daten Gespeichert werden müssen die Daten der Elemente des Szenegraphen und die Relationen (Pfeile). Um beide Speicherformate zu unterstützen, müssen für beide Methoden vorhanden sein. Den Dateipfad legt der Benutzer vorher mit Hilfe eines Speicherdialogs fest. Der Dateiname und eine Referenz auf den Controller der Anwendung müssen den Speichermethoden als Parameter übergeben werden. Die Methoden rufen die zu speichernden Daten direkt beim Controller ab und speichern sie in einer Datei mit dem übergebenen Pfad.

Zusätzlich zu diesen Daten ist es erforderlich, dass die Höhe und die Breite des Zeichenpanels gespeichert werden. Dies muss deshalb geschehen, damit beim Laden Scrollbalken angezeigt werden, falls die Zeichenfläche größer als der im Scrollbereich darstellbare Platz ist. Die Alternative dazu wäre, alle Elemente des Szenegraphen zu durchsuchen, und die Höhe und Breite nach dem jeweils am weitesten von der linken oberen Ecke entferntem Element zu ermitteln.

Laden der Daten Die beiden Methoden zum Laden der Daten erfordern nur einen Dateipfad, den der Benutzer ebenfalls per Dialog auswählt. Dabei werden die oben in eine Datei gesicherten Daten wieder in den Controller geladen, sei es nun direkt als serialisierte Objekte oder per Umweg über einen XML-Parser.

Zu diesem Zeitpunkt existieren aber nur die Daten der Elemente, ihre Repräsentationen auf der Zeichenfläche fehlen noch. Diese sind Instanzen einer nur zur Oberflä-

chendarstellung benötigten und von Panel abgeleiteten Klasse und müssen erst wieder neu initialisiert werden. Dazu muss für jedes einzelne Datenobjekt ein Panel mit dessen Typ erstellt werden. An dieser Stelle müssen dem Panel auch seine die Oberfläche betreffenden Daten übermittelt werden. Nur dann ist gewährleistet, dass der Szenegraph genauso wie vor dem Speichern dargestellt wird.

Weiterhin müssen initiale Angaben gesetzt werden. Die wichtigste Angabe ist die ID. Sie ist innerhalb eines Szenegraphen einzigartig. Da die Daten des Szenegraphen in der Liste nicht umsortiert werden, ist das letzte Datenobjekt immer das mit der höchsten ID. Diese wird beim Laden um eins inkrementiert und ist dann die neue ID für das nächste Element, welches in den Szenegraph eingefügt wird.

Als weitere Aktionen müssen die Farbe für neue Elemente auf Schwarz gesetzt, die geladene Datei als gespeichert markiert, die Listen mit den markierten Elementen geleert und alle vorher laufenden Aktionen abgebrochen werden. Außerdem werden alle temporären Angaben auf ihre Standardwerte eingestellt.

3.1.3. Semantik eines Szenegraphen

Damit der Szenegraph erfolgreich in Quellcode umgesetzt werden kann, muss er gewissen formalen Regeln entsprechen. Diese werden in folgendem Abschnitt zu einem Regelwerk zusammengefasst. Es soll festlegen, welche Knoten und Knotenkomponenten miteinander verbunden werden können. Für das Verbinden gibt es, wie bereits mehrfach erwähnt, zwei Relationen, die Vater-Kind-Beziehung und die Referenz.

- Vater-Kind-Beziehung
  - Vater einer VKB kann nur ein Gruppenknoten sein.
  - Kind einer VKB können Gruppen- oder Blattknoten sein.
  - Gruppenknoten haben maximal einen Vater.
  - Blattknoten haben maximal einen Vater.
  - Gruppenknoten können keine bis unendlich viele Kinder besitzen.
  - Blattknoten besitzen keine Kinder.

- Referenz
  - Startelement einer Referenz kann ein Gruppen-, Blattknoten oder eine Knotenkomponente sein.
  - Endelement einer Referenz kann ein Gruppen-, Blattknoten oder eine Knotenkomponente sein.
  - Ein Element kann auch mehrfaches Endelement einer Referenz sein.
  - Ein Element kann Startelement mehrerer Referenzen sein.

Daneben existieren bei einigen 3D-Klassen Ausnahmen; sowohl für Vater-Kind-Beziehungen als auch für Referenzen.

Bei einer Referenz ist nicht per Definition ersichtlich, wer mit wem verbunden werden darf. Vielmehr erschließt sich dies erst beim Studium der API und Umsetzung in eine Codegenerator-Klasse. Deshalb muss das Datenobjekt eines Elements eine Liste mit den IDs der Elemente enthalten, die als Endelement einer Referenz vom aktuellen Element in Frage kommen. Sie ist für jede Klasse bereits beim Erzeugen definiert und kann vom Benutzer nicht geändert werden. Beim Anlegen einer Referenz wird dieses Liste abgefragt. Wenn eine ID enthalten ist, wird per Klick auf das Element eine neue temporäre Referenz erzeugt. Fährt der Benutzer mit der Maus über ein anderes Element, so wird erneut auf die Liste zugegriffen. Nur wenn die ID des Elements enthalten ist, kann per Klick die Referenz erzeugt werden.

Diese Maßnahme ist für den Benutzer nützlich, denn so kann er nur erlaubte Referenzen anlegen. Allerdings muss gewährleistet sein, dass alle benötigten Referenzen in der Liste enthalten sind, da dies nur durch ein erneutes Kompilieren des Codegenerators korrigiert werden kann.

Als zusätzliche Hilfe kann der Benutzer im Optionsdialog einstellen, ob er auf fehlende Referenzen sofort hingewiesen werden möchte. Standardmäßig geschieht dies nur beim Erstellen des Quellcodes. Setzt der Benutzer diese Option, wird bei jedem Erstellen oder Löschen einer Referenz eine Prüfung durchgeführt, ob alle Elemente die von ihnen zum korrekten Funktionieren benötigten Referenzen besitzen. Ist dies nicht der Fall, wird der Name des Elementes rot markiert. Will der Benutzer den Quellcode erstellen und es fehlen noch Referenzen, erscheint zusätzlich ein Informationsdialog.

Anders als bei Referenzen gibt es bei einer Vater-Kind-Beziehung weniger Ausnahmen, bei denen nur bestimmte Knoten miteinander verbunden werden können. Auch weisen die meisten Knoten genau einen Vaterknoten auf; nur in einigen Fällen besitzt ein Knoten keinen Vater.

Zu beiden Kategorien zählt die Klasse "SharedGroup". Sie bietet die Möglichkeit, einen Graphen zu kompilieren und mit Hilfe von "Link"-Knoten mehrfach im Szenegraphen zu verwenden. Er besitzt keinen Vater und es dürfen nur bestimmte Knoten eingefügt werden. Deshalb enthält das Datenobjekt eine weitere Liste mit den IDs der erlaubten Kindsknoten. Sind keine Einschränkungen vorhanden, so bleibt diese Liste leer. Auch hier erfolgt beim Anlegen eine Prüfung, ob der Kindsknoten ein zulässiger ist.

3.1.4. Oberfläche des Codegenerators

Die Oberfläche wird mit Java Swing erstellt und beschränkt sich im Wesentlichen auf das Hauptfenster, also die Klasse "FrmMain" sowie weitere Klassen, die die Darstellung von weiteren Dialogfeldern und Oberflächenkomponenten ermöglichen. Da die wichtigsten Funktionen sich jedoch im Hauptfenster befinden, soll dieses im Folgenden zuerst erläutert werden.

Hauptfenster Das Hauptfenster der Anwendung besteht aus vier Teilbereichen: Menüleiste, Toolbarbereich, Zeichenfläche und Statusleiste, welche in den nächsten Abschnitten genauer erläutert werden.

Menüleiste Über die Menüleiste kann der Benutzer alle wichtigen Funktionen des Programms erreichen, wobei die wichtigsten Menüpunkte mit standardisierten Symbolen gekennzeichnet werden und die allgemeingültigen Tastaturkürzel gelten. Im Menü "Datei" kann er einen Szenegraphen anlegen, laden und speichern. Daneben bietet es die Möglichkeit, den aktuell angezeigten Szenegraphen in eine PNG-Datei zu exportieren. In diesem Menü kann das Programm auch beendet werden.

Das nächste Menü ist modulspezifisch und muss beim Laden des Moduls in die Menüzeile eingefügt werden. Darüber kann ein Element in einen Szenegraphen einge-

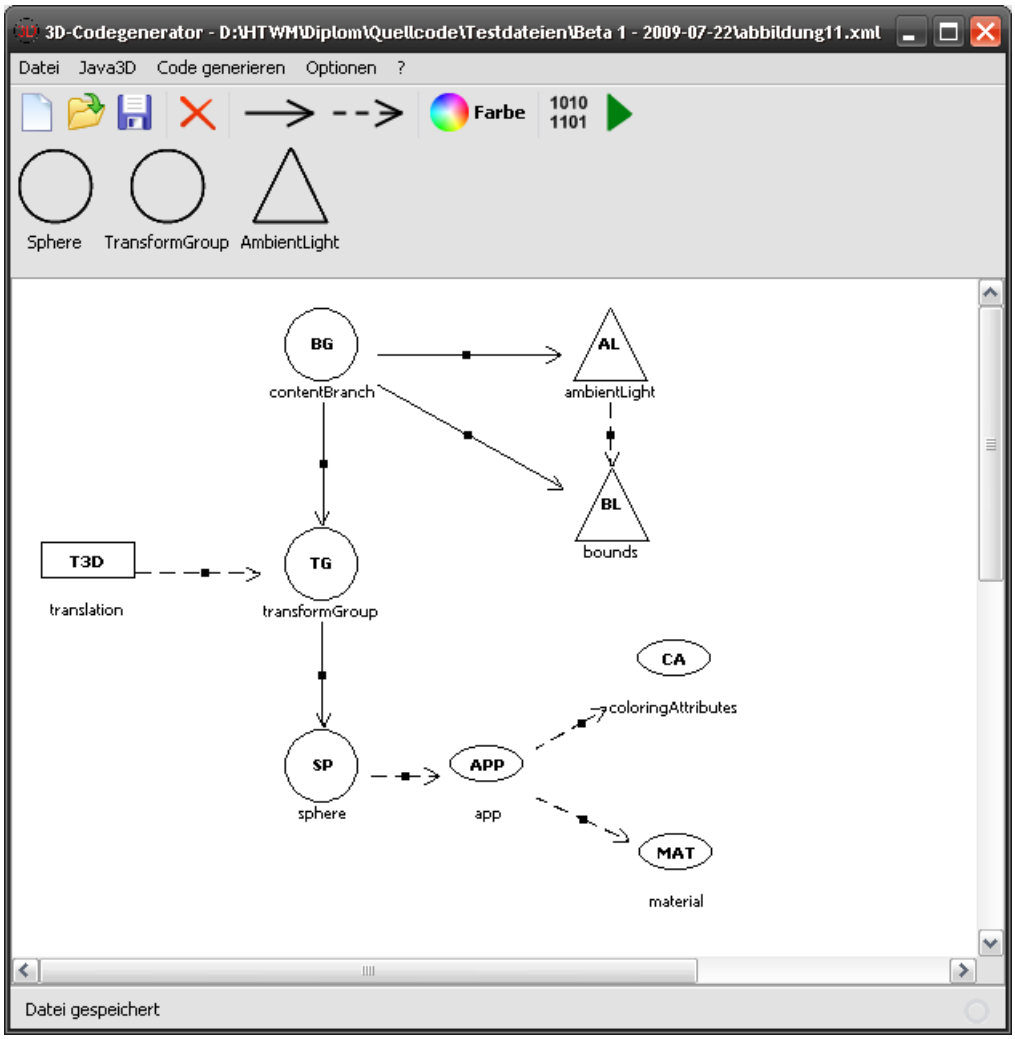


Abbildung 3.1.: Hauptfenster der Anwendung

fügt werden, in der Abbildung ist das Java3D-Menü zu sehen. Es ist in drei Untermenüs gegliedert, welche jeweils alle Elemente von Gruppenknoten, Blattknoten und Knotenkomponenten enthalten. Dabei werden inhaltlich ähnliche Elemente in weiteren Untermenüs gruppiert. Jeder Menüpunkt ist außerdem zur einfachen Einordnung noch mit dem Symbol seines Typs versehen, also einem Kreis für einen Gruppen- oder Dreieck für einen Blattknoten. Knotenkomponenten werden entweder durch eine Ellipse oder ein gemischtes Symbol dargestellt, da bei einigen Knotenkomponenten am Anfang noch nicht feststeht, ob sie im konkreten Fall erforderlich oder optional sind.

Das nächste Menü enthält Befehle, die das Erstellen von Quellcode und das anschließende Starten des Programms ermöglichen. Der Menüpunkt ”Quellcode erzeugen“ ge-

neriert nur den Code, startet aber kein weiteres Programm. Er eignet sich, um schnell zu überprüfen, ob der Szenegraph in seiner jetzigen Form kompilierbar ist. Der zweite Menüpunkt erstellt ebenfalls den Quellcode und übersetzt diesen anschließend durch Aufruf des "javac"-Programms in Java-Byte-Code. Sollte hierbei ein Fehler festgestellt werden, so wird eine Fehlermeldung ausgegeben. Danach wird das generierte Programm gestartet und läuft dann als eigenständiger Prozess. Der Benutzer kann es jederzeit per Klick auf "Schließen" beenden.

Im später näher erläuterten Optionsdialog kann der Benutzer verschiedene Einstellungen durchführen.

Im letzten Menü kann ein Informationsdialog über das Programm geöffnet werden. Hier ist auch Platz für eine eventuelle spätere Programmhilfe.

Allgemeine Toolbar Der nächste Bereich ist die allgemeine Toolbar, welche direkt unter der Menüleiste angeordnet und durch Separatoren in mehrere Gruppen unterteilt ist. Die erste Gruppe enthält wie in den meisten Programmen einige Buttons, die einen Schnellzugriff auf Aufgaben im "Datei"-Menü ermöglichen.

Die nächste Gruppe beinhaltet nur einen Button, den zum Löschen einer oder mehrerer Komponenten des Szenegraphen. Diese Funktion ist auch erreichbar, wenn die "Entf"-Taste gedrückt wird, während Komponenten markiert sind. Die oberste BranchGroup, also die Wurzel des Szenegraphen, kann allerdings nicht gelöscht werden.

Danach folgen die Buttons zur Erzeugung von einer Vater-Kind-Beziehung und einer Referenz, jeweils durch einen Pfeil symbolisiert. Dieser wird bei ersterer durchgezogen und bei letzterer gestrichelt dargestellt.

Die nächste Gruppe enthält ebenso nur einen Button. Über diesen ist es möglich, die Farbe für neue und bestehende Elemente zu setzen. Beim Klick darauf wird ein Fenster geöffnet, in dem der Benutzer mit Hilfe eines JColorChoosers eine Farbe auswählen kann. Wird das Fenster per "OK" geschlossen, so wird die Farbe für die nächste Komponente auf die vom Benutzer gewählte gesetzt. Sind Komponenten markiert, wird deren Farbe ebenfalls geändert. Beim Speichern wird die Farbe für die nächste Komponente nicht mit gespeichert. Wird die Datei demnach neu geladen, so steht die Farbe wieder auf Schwarz.

In der letzten Gruppe sind Buttons enthalten, mit deren Hilfe der Quelltext erstellt sowie kompiliert und das erstellte Programm gestartet werden kann. Sie besitzen die gleiche Funktion wie im Menü ”Code generieren“.

Alle Buttons zeigen einen Hilfetext als ToolTip an, wenn der Mauszeiger länger über ihnen verweilt.

Toolbar mit den zuletzt genutzten Elementen Die Toolbar enthält Buttons, die die am häufigsten im Szenegraph verwendeten Elemente repräsentieren. Klickt der Benutzer auf sie, kann er per Mausklick ein weiteres Element auf der Zeichenfläche einfügen. Diese von ”JButton“ abgeleitete Klasse benötigt dazu beim Initialisieren eine Typ-ID und eine Klassen-ID. Sie wird erzeugt, wenn der Benutzer auf einen Eintrag im modulspezifischen Menü klickt.

Je nach Typ wird als Symbol entweder ein Kreis, ein Dreieck oder eine Ellipse verwendet. Damit die Toolbar nur die Buttons der am meisten benutzten Elemente enthält, müssen die Buttons eine Angabe beinhalten, wie oft sie bereits angeklickt wurden. Standardmäßig wird diese nach dem Anlegen mit Eins initialisiert.

Damit sich nach dem Einfügen eines neuen Buttons nicht zwei Buttons des gleichen Typs in der Menüleiste befinden, muss nach dem Klick auf einen Menüeintrag geprüft werden, ob sich in der Toolbar bereits ein Button mit der gleichen Klassen-ID wie der gerade angeklickte Menüeintrag befindet. Genau aus diesem Grund muss die Routine zum Anlegen eines neuen Elements durch den Button ausgeführt werden. Dazu bietet die abstrakte Klasse ”AbstractButton“ mit ”doClick()“ eine spezielle Methode, mit der sich ein Mausklick auslösen lässt. Diese ist durch die Vererbung auch in der Klasse JButton verfügbar.

Befindet sich in der Menüleiste noch kein Button mit der gleichen Klassen-ID, so wird in dieser Toolbar ein Button angelegt und automatisch per ”doClick()“ angeklickt. Dadurch kann mit einem weiteren Mausklick auf die Zeichenfläche der gewählte Knoten oder die Knotenkomponente angelegt werden. Befindet sich in der Menüleiste bereits ein Button mit der gleichen Klassen-ID, so wird dessen interner Zähler um eins inkrementiert und wieder alles zum Anlegen eines neuen Elements vorbereitet.

Als Vorbereitungsaktionen werden ein neues Datenobjekt vom gewählten Klassentyp

und ein dazu passendes Panel für die Zeichenfläche angelegt. Wird diese angeklickt, so wird das Datenobjekt dem Vektor mit den Datenobjekten hinzugefügt und das Panel der Zeichenfläche am Punkt des Mauszeigers.

In die Toolbar passt nur eine bestimmte Anzahl von Buttons. Ist die maximale Anzahl erreicht, so wird der Button, der am wenigsten angeklickt wurde, aus der Toolbar entfernt und durch einen neuen ersetzt, welcher das vom Benutzer zuletzt im Menü angeklickte Element erzeugt.

Zeichenfläche Die Zeichenfläche soll sich für den Benutzer als weiße Fläche darstellen. Auf dieser wird der Szenegraph angelegt, indem aus dem Menü Elemente eingefügt und mit Pfeilen verbunden werden. Die Elemente sind von Panel abgeleitete Klassen, die Pfeile werden wiederum nur gezeichnet. Mit diesen Voraussetzungen bietet sich ein Panel als Zeichenfläche an. Allerdings muss dafür eine eigene Klasse erstellt werden, die von Panel erbt. Die Zeichenfläche enthält zusätzliche Attribute und Methoden; auch werden die Ereignisroutinen durch eigene Methoden ersetzt.

Da die Pfeile direkt auf der Zeichenfläche gezeichnet werden, muss diese nach jeder vom Benutzer ausgelösten Aktion, bei der Elemente verschoben oder gelöscht werden, neu gezeichnet werden. In der dazu notwendigen Zeichenmethode werden nun die einzelnen Teilabschnitte des Pfeiles gezeichnet, beginnend beim Start- bis zum Endpunkt; wobei die Farbe und die Linienform des Pfeiles berücksichtigt werden.

Diese Zeichenmethode zeichnet auch ein Auswahlrechteck auf der Zeichenfläche, wenn der Benutzer mehrere Elemente markieren will.

Statusleiste In dieser Leiste wird bei verschiedenen Aktionen deren Status angezeigt. Der Benutzer wird so dezent über aktuelle Aktionen informiert, für die sonst ein Informationsdialog verwendet wird.

Dies sind die vier Teilbereiche des Hauptfensters. Im Folgenden werden weitere Elemente der grafischen Oberfläche vorgestellt.

Optionsmenü Das Optionsmenü bietet die Möglichkeit, verschiedene Programmoptionen zu ändern. Zur einfachen Speicherung bietet sich eine im gleichen Programm-

verzeichnis liegende Textdatei an. Der Benutzer kann die Einstellungen sowohl direkt in dieser vornehmen oder dazu das Optionsmenü nutzen, welches per Klick auf den gleichnamigen Menüeintrag erscheint.

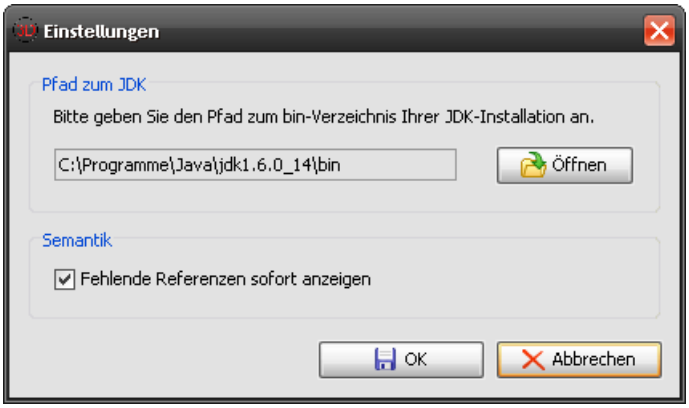


Abbildung 3.2.: Optionsdialog

Als erste und vielleicht wichtigste Einstellung kann hier der Pfad zu einem JDK eingetragen werden. Bei jedem Kompilieren muss das Programm testen, ob in dem angegebenen Ordner eine Datei namens ”javac“ vorhanden ist. Falls nicht, wird eine Fehlermeldung ausgegeben und der Vorgang abgebrochen. Der Benutzer bekommt spätestens jetzt einen Hinweis darauf, wie er den Pfad einstellen kann.

Weiterhin kann der Benutzer entscheiden, wann er über fehlende Referenzen unterrichtet werden möchte.

Eigenschaften eines Elements Die folgende Abbildung zeigt das Eigenschaftsfenster eines Knoten oder einer Knotenkomponente.

Im Beispiel sind die Eigenschaften einer ”Material“-Knotenkomponente dargestellt. Im oberen Teil sind die allgemeinen Angaben angeordnet, der Name und die Farbe des Elements. Darunter ist der Bereich, in dem die verschiedenen Attribute untereinander angezeigt werden. Ein Klick auf den Button ”Farbe“ ermöglicht es dem Benutzer, die einzelnen Farben zu setzen. Passen die Variablen nicht in den dafür vorgesehen Bereich, so wird automatisch eine Scrollleiste angezeigt.

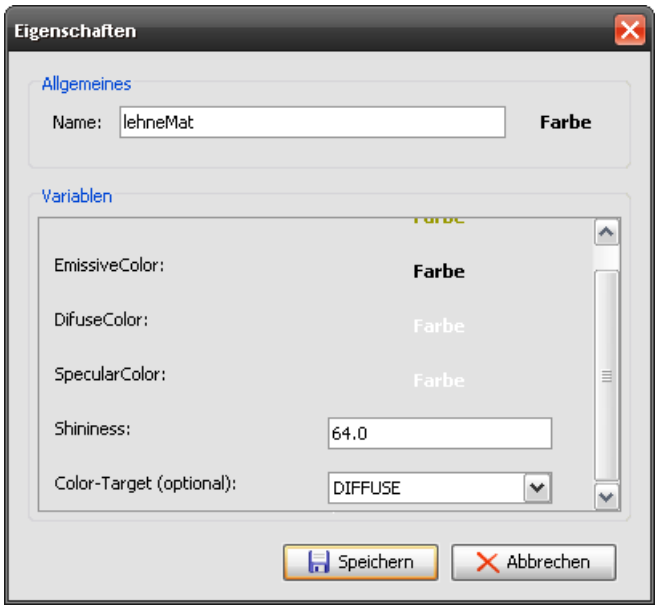


Abbildung 3.3.: Eigenschaftsdialog eines Elements

Eigenschaften eines Pfeils Die folgende Abbildung zeigt das Eigenschaftsfenster einer Vater-Kind-Beziehung oder einer Referenz, welches für beide die gleichen Einstellmöglichkeiten bietet. Die Art der Beziehung kann hier nicht mehr geändert werden.

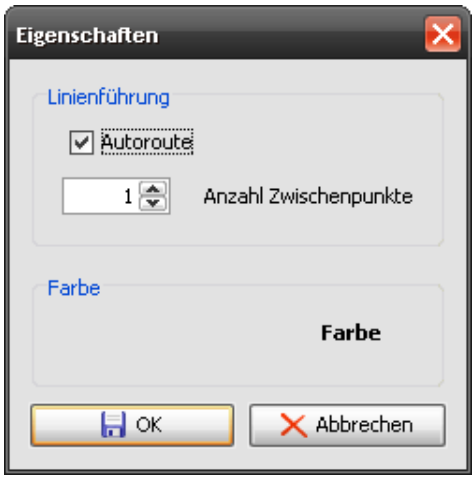


Abbildung 3.4.: Eigenschaftsdialog einer Relation

Darstellung von Elementen Für die Darstellung der Elemente wird ebenfalls eine von Panel abgeleitete Klasse verwendet. Sie stellt für die Interaktionen vom Benutzer mit dem Element verschiedene Attribute und Methoden zur Verfügung. Da es drei

Typen von Elementen gibt, die sich in der Darstellung unterscheiden, erhält jeder dieser Typen eine eigene Klasse. Sie stellt allerdings lediglich einen Konstruktor und eine Methode zum Zeichnen bereit. Die anderen Funktionen erbt sie von einer für alle Typen gleichen zentralen Klasse.

Sie besitzt eine Referenz auf ihr Datenobjekt und stellt dieses per Methodenaufruf zur Verfügung. Dabei reicht sie einige Methoden an dieses weiter. Beispiele dafür sind ”getID()“ oder ”getColor()“.

Daneben enthält sie Methoden, die das Verhalten auf der Zeichenoberfläche steuern, damit der Benutzer das Programm wie jedes andere bedienen kann. Dazu zählen die Methoden, welche den Knoten oder die Knotenkomponente entweder selektieren, oder deselektieren. Im selektierten Modus kann das Element vom Benutzer bearbeitet werden.

Die Klasse prüft außerdem laut dem Regelwerk, ob eine Vater-Kind-Beziehung oder eine Referenz hergestellt werden kann. Dazu werden zwei Prüfungen durchgeführt, die erste bereits, wenn der Benutzer mit der Maus über ein Element fährt und vorher in der Toolbar einen Button zum Anlegen einer Beziehung angeklickt hat. Fährt er mit der Maus über eine gültige Start- oder Endposition, verändert sich der Mauszeiger und zeigt einen je nach Relation verschiedenen Pfeil mit einem grünen Haken. Befindet sich der Mauszeiger hingegen über einem ungültigen Element oder auf freier Fläche, so wird aus dem grünen Haken ein rotes Kreuz. So bekommt der Benutzer bereits hier einen Hinweis, ob die Relation angelegt werden kann. Klickt er nun auf ein Element, wird nochmals geprüft, ob die Verbindung hergestellt werden kann. Dies erfolgt mit der gleichen Vorgehensweise.

Weiterhin steuert die Klasse alle Vorgänge nach dem Verschieben eines Elements, bei dem die Maus über einem Element gedrückt gehalten und das Element an die Position verschoben wird, an der sich der Mauszeiger gerade befindet. Dabei muss beachtet werden, dass der Mauszeiger und die Zeichenfläche nicht das gleiche Koordinatensystem besitzen. Es muss also eine Umrechnung stattfinden, welche der Controller bereitstellt. Weiterhin ist es wichtig, bereits vor dem Verschieben den Abstand des Mauszeigers zur linken oberen Ecke des Elements zu bestimmen, da sonst bei der Durchführung der Aktion der Mauszeiger immer genau auf dieser positioniert wird. Dies wird in der

Ereignisroutine erledigt, wenn der Mauszeiger gedrückt wird.

Nach dem eigentlichen Verschieben müssen weitere Vorgänge ausgeführt werden. Als Erstes sei hier das Vergrößern der Zeichenfläche zu nennen. Diese ist standardmäßig immer so groß wie der sie beinhaltende Container. Wird ein Element über den sichtbaren Rand verschoben, so muss sie vergrößert werden, damit Scrollbalken angezeigt werden. Außerdem müssen nach jeder Verschiebung andere eventuell ebenso markierte Elemente neu platziert werden. Bei einer Mehrfachauswahl müssen alle in die gleiche Richtung verschoben werden, um die individuellen Abstände zum vom Benutzer bewegten Element nicht zu verändern. Sind alle Positionen geändert, sind natürlich die mit ihnen verbundenen Pfeile zu aktualisieren. Dies wird für jedes Element ausgeführt, welches markiert ist.

Erst wenn all diese Aktionen ausgeführt sind, kann die Zeichenoberfläche neu gezeichnet werden.

3.1.5. Generierung von Quellcode

Wie oben erwähnt, generiert jedes Elements seinen Quellcode selbst. Allerdings wird ein Einstiegspunkt benötigt, welcher die Erstellung von Quellcode im ersten Element startet und die noch fehlenden statischen Quellcodefragmente hinzufügt.

Es bietet sich an, eine dafür eigens angelegte Klasse zu verwenden, die diese Aufgaben in Form von statischen Methoden erledigt. Damit sie Zugriff auf die Daten erhält, benötigt sie eine Referenz auf die Liste mit den Daten im Controller. Als Name der generierten Klasse wird jene Bezeichnung genutzt, die der Benutzer zum Speichern der Szenegraph-Datei verwendet hat. Dabei wird der Name eventuell an die Java-Namenskonvention angepasst. Ist der Szenegraph noch nicht gespeichert, so muss ein Name generiert werden, zum Beispiel ”UnSavedClass“.

Beim Generieren des Quellcodes müssen zuerst die statischen Anteile erstellt werden, ohne welche der erstellte Quellcode nicht kompilierbar wäre.

Als erstes sind hier die Importe zu nennen. Standardmäßig werden alle Pakete von Java3D importiert, mit Ausnahme von Paketen, die als ”deprecated“ oder ”obsolete“ gekennzeichnet sind. Ungenutzte Pakete werden vom Compiler beim Kompilieren erkannt

und ausgefiltert. Der Benutzer kann diese jederzeit gefahrlos löschen.

Nach den Importen folgt im Quelltext die eigentliche Klasse, die wiederum aus mehreren statischen Teilen besteht. Dies sind Konstruktor und main()-Methode. Beide sind Grundvoraussetzung dafür, dass das Programm funktioniert, da sie gewisse initiale Eigenschaften setzen; wie zum Beispiel eine Ereignisbehandlungsroutine zum Schließen des Programms.

Die eigentliche Quelltexterzeugung wird von einer eigenen Methode ausgeführt. Sie beginnt die Quelltexterzeugung, indem sie lediglich den Rumpf der Methode und das erste Element, also die initiale BranchGroup erzeugt. Deshalb muss diese immer vorhanden sein und kann vom Benutzer nicht gelöscht werden, denn ohne sie würde der Methode der Einstiegspunkt fehlen.

Von hier ab geschieht die Quelltextgenerierung rekursiv, das bedeutet, dass der erste Kindsknoten der BranchGroup erzeugt wird. Hat dieser selbst Kinder, werden auch diese nacheinander erzeugt. Ist das Ende eines Zweiges erreicht, also ein Blatt- oder ein Gruppenknoten ohne Kinder, so wird auf der letzten Ebene mit dem nächsten Kindsknoten fortgefahren. Die Erzeugung des individuellen Quellcodes der Elemente übernimmt deren eigene Methode, die als Parameter die Anzahl der dem Code voranzustellenden Tabulatoren und eine Referenz auf den Controller erfordert.

Neben dem oben genannten rekursiven Vorgehen muss eine weitere Besonderheit beachtet werden, die Referenzen betreffend. Anders als bei Gruppen- oder Blattknoten, die jeweils nur einen Vater haben können, können Referenzen auf mehrere andere Elemente referenzieren, oder von anderen Elementen referenziert werden. Somit könnte die Situation entstehen, dass ein Element beim Erzeugen ein Referenzobjekt erzeugt, welches bereits generiert wurde. Damit wäre eine Variable doppelt vorhanden, wodurch der Quellcode nicht kompilierbar ist.

Deshalb muss gewährleistet sein, dass vor dem Erzeugen geprüft wird, ob ein Element bereits im Quelltext generiert wurde. Ist dies der Fall, wird die Methode beendet. Ansonsten wird als erste Anweisung die Angabe gesetzt, dass der Quellcode erzeugt wurde, eben um oben genanntem Problem vorzubeugen. Anschließend wird der Quelltext für den Konstruktor eingefügt, sowie die Anweisungen zum Einbinden von allen Attributen. Diese können auf zwei Arten in den Quelltext eingebunden werden: Per

Konstruktor oder per Methode.

Da die meisten Eigenschaften der Elemente per Methodenzugriff gesetzt werden, erfolgt auch die Übergabe von Parametern in der Regel über einen Methodenaufruf. Verlangt der Konstruktor zwingend nach einem Parameter, kann meistens auch NULL übergeben werden. In einigen Fällen erzeugt dieses Vorgehen jedoch eine NullPointerException, dann muss dieser Parameter bereits vor dem Konstruktor erzeugt und diesem übergeben werden.

Welche Parameter dieses Verhalten zeigen, ist durch Studium der API zu erfahren. Manchmal kann es aber nur durch Testen der Klasse herausgefunden werden. Ein Beispiel ist der SwitchValueInterpolator, welchem das Switch-Target bereits im Konstruktor übergeben werden muss. Bei anderen Interpolatoren ist dies nicht der Fall.

Danach werden je nach Typ des Elements unterschiedliche Schritte ausgeführt. Ist das Element ein Gruppenknoten, dann werden zuerst die Kinder des Knoten erzeugt. Dazu werden die in den Daten gespeicherten Kinder nacheinander generiert, falls sie nicht bereits existieren. Anschließend wird das Kind durch Hinzufügen der Anweisung ”addChild“ im Gruppenknoten als Kind angefügt.

Je nachdem, ob das Element Start- oder Endelement einer Referenz sein kann, müssen nun die Referenzobjekte eingebunden werden. Dabei werden beide Listen mit den Referenzobjekten durchlaufen und jedes Einzelne wird, falls noch nicht erzeugt, erstellt. Danach muss das Referenzobjekt in den Quelltext eingebunden werden. Dies kann sowohl im Start- als auch im Endelement der Referenz erfolgen. Damit die Übersichtlichkeit gewahrt bleibt, geschieht dies allerdings einheitlich im Startelement.

Da je nach Klasse die Methode einen anderen Namen besitzen kann, wird eine Unterscheidung durchgeführt. Als Beispiel sei hier die ”Transform3D“-Knotenkomponente genannt. Die Einbindung erfolgt je nach Art der Klasse; in einer TransformGroup mit der Methode ”setTransform(Transform3D t3d)“, in der Klasse RotationInterpolator per ”setTransformAxis(Transform3D t3d)“. Deswegen ist es auch nötig, jedem Element beim Initialisieren mitzuteilen, mit welchen anderen Elementen es eine Referenz eingehen kann.

Ist der Quelltext von den Methoden komplett erzeugt, wird er zu einem Text zusammengefügt und in eine Datei gespeichert. Diese liegt im gleichen Verzeichnis wie

die Datei, die den Szenegraphen enthält. Im Falle eines ungespeicherten Szenegraphen liegt sie im HOME-Verzeichnis des Benutzers. Die Methode zum Speichern kann die für das Speichern und Laden eines Szenegraphen zuständige Klasse übernehmen.

3.2. System-Struktur

Das System wird grundsätzlich so erstellt, dass die drei Ebenen Oberfläche, Controller und Datenhaltung getrennt und im Programm in verschiedene Pakete eingeteilt werden.

Das folgende Klassendiagramm zeigt den allgemeinen Aufbau des Programms mit seinen Basisklassen. Dabei ist die Trennung in die einzelnen Schichten deutlich zu sehen, welche sich auch in den im Paket "modules" liegenden Modulpaketen vollzieht.

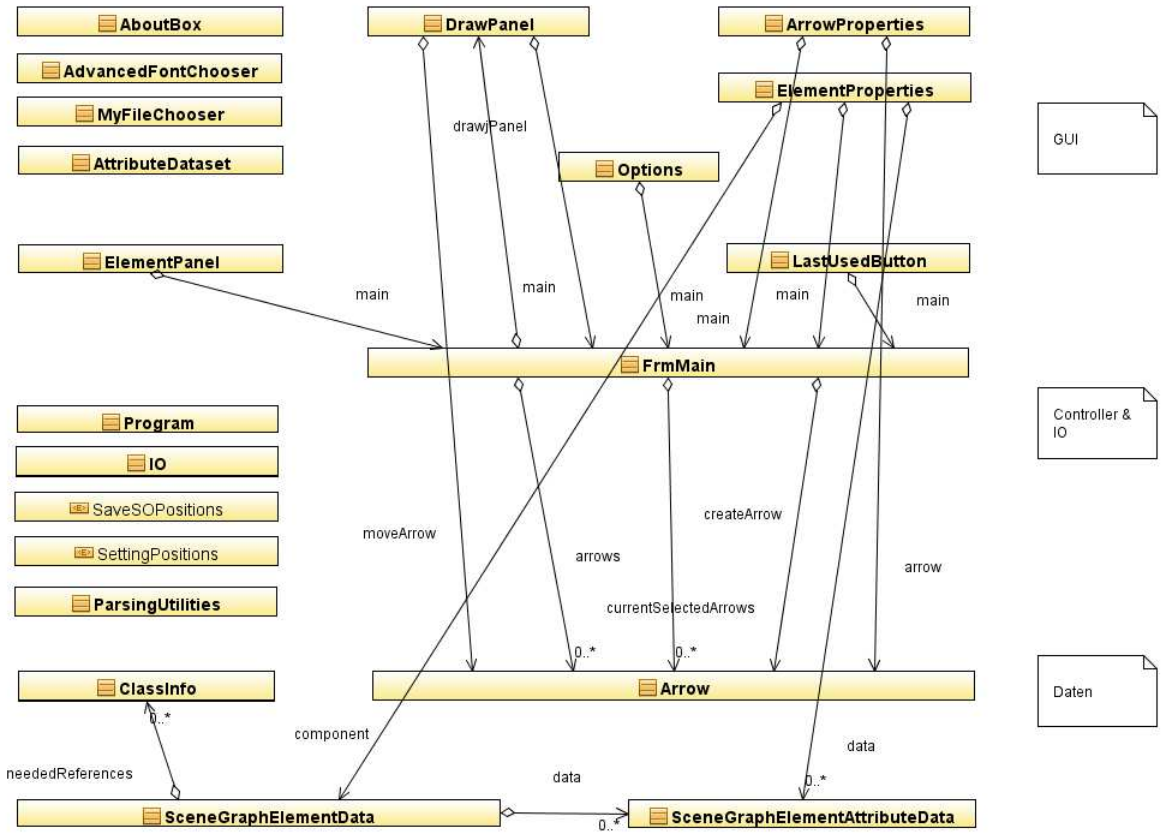


Abbildung 3.5.: Allgemeines Klassendiagramm

Die Paketnamen orientieren sich an ihrer jeweiligen Schicht. Hinzu kommt noch das Paket "constants", welches für den Betrieb des Codegenerators wichtige Konstanten

enthält. Diese sind zur Vereinfachung in jenen Klassen zusammengefasst. Im Diagramm sind dessen Klassen mit in der Controllerschicht enthalten.

Die Klasse ”FrmMain“ nimmt dabei eine Sonderstellung ein. Sie erbt von der Klasse ”FrameView“ und stellt wie aus ihrem Namen ersichtlich, eine Sicht auf ein Fenster bereit. In diesem Falle ist das das Hauptfenster der Anwendung, welches in der Klasse selbst durch Aufruf der Methode ”this.getFrame()“ erreichbar ist. Daneben stellt die ”FrmMain“ weiterhin typische Funktionen eines Controllers bereit. Sie verwaltet den Datenbestand, enthält Statusvariablen und temporäre Daten. Daneben bietet sie Zugriff auf Funktionen der Klasse ”IO“. Aufgrund ihrer Sonderstellung steht sie im Diagramm zwischen den Schichten Oberfläche und Controller.

Im Folgenden werden die drei Schichten mit ihren Klassen vorgestellt.

Controller / ”controller“ Das Paket enthält nur die Klasse ”IO“, welche statische Methoden zum Speichern und Laden von einem Szenegraphen und von Einstellungen aus einer Datei enthält. Sie selbst hält dabei keine Variablen zur Abfrage bereit. Außerdem bietet sie eine Methode an, mit der der erzeugte Quellcode in einer Datei gespeichert wird.

Die eigentliche Controllerklasse ”FrmMain“, die auch das Hauptfenster darstellt, ist wegen eben dieser Darstellung im Paket “gui“ angesiedelt. Sie beinhaltet mit den Vektoren ”sceneGraphElementsData“ und ”arrows“ die eigentlichen Daten des Szenegraphen und stellt diese den anderen Klassen und Oberflächenkomponenten zur Verfügung. Weiterhin enthält sie Hilfsangaben und temporäre Variablen.

Außerdem beherbergt das Paket die Klasse ”Program“, welche die Main-Methode beinhaltet, durch die der Codegenerator gestartet wird.

Konstanten / ”constants“ Dieses Paket enthält die Enumerationen ”SaveSOPositions“ und ”SettingPositions“. Sie beinhalten die Reihenfolgen, die bei der Speicherung von einem Szenegraphen in Form von serialisierten Objekten sowie der Einstellungen eingehalten werden müssen. Diese ist wichtig, da nur Daten in der korrekten Reihenfolge eingelesen werden können.

Die Einstellungen des Codegenerators werden aus einer Textdatei in einen Vektor geladen und zwar jede an die in "SettingPositions" angegebene Position. Die Angabe, ob fehlende Referenzen sofort geladen werden sollen, lädt der Controller von der in der Enumeration angegebenen Position.

Das Paket beinhaltet außerdem die Klasse "ParsingUtilities". Sie stellt für jeden als Eingabevariable benötigten Datentyp Methoden zur Verfügung, um diesen in einen String umzuwandeln und aus einem solchen parsen zu können. Dabei werden auch Methoden für Datentypen bereitgestellt, die solche selbst enthalten; zum Beispiel int. Die selbst definierten Methoden bieten jedoch den Vorteil, dass sie bereits eine Fehlerbehandlung inne haben.

Datenhaltung / "data" Dieses Paket enthält alle Klassen und Elemente, die die Daten eines Szenegraphen enthalten. Als Erstes sind hier die eigentlichen Datenklassen eines Szenegraphen zu nennen, "SceneGraphElementData", "SceneGraphElementAttributeData" und "Arrow". Sie stehen allen anderen Modulen zur Verfügung. Daneben existiert mit der Klasse "ClassInfo" noch eine Hilfsklasse, die die ID, den Namen einer 3D-Klasse und die maximale Anzahl von Objekten dieser Klasse für weitere Anwendung bereitstellt.

Oberfläche / "gui" Das Paket "gui" enthält alle Klassen zur Darstellung der Oberfläche. Darunter zählen neben dem Hauptfenster in der Klasse "FrmMain" und den dazugehörigen Klassen wie "AboutBox" und anderen auch die Zeichenfläche "DrawPanel" sowie die allgemeinste Darstellung von Elementen auf dieser: "ElementPanel". Des Weiteren sind die Eigenschaftsfenster von Elementen, "ElementProperties" und "ArrowProperties", enthalten.

Module / "modules" In diesem Paket sind weitere Pakete enthalten, die jeweils Klassen für ein Modul beinhalten. Sie erweitern dabei die drei Ebenen um weitere für sie spezifische Klassen.

Das folgende Diagramm zeigt den Aufbau des Pakets "java3d", dessen Klassen immer mit dem Präfix "CGJ3D" beginnen.

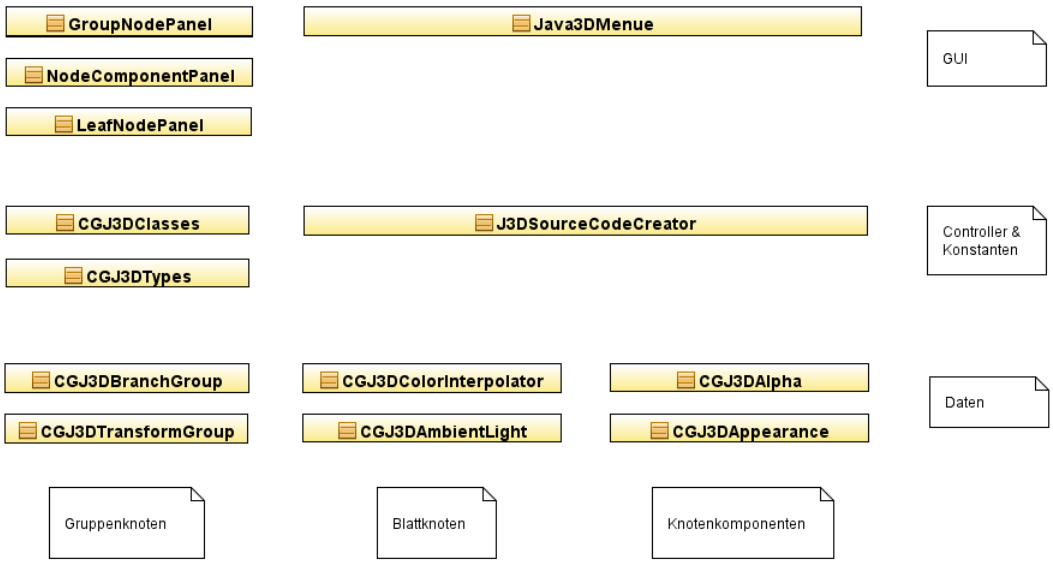


Abbildung 3.6.: Klassendiagramm des Java3D-Moduls

Es erweitert die Datenebene für Java3D-Klasse um eine eigene Klasse, die von ”SceneGraphElementData“ erbt. Sie stellt weitere für die Klasse spezifische Daten bereit und enthält mit der Methode ”GenerateCode()“ die eigentliche Methode zur Erzeugung von Quellcode. Die einzelnen Klassen sind je nach Typ nochmals in Unterpaketen gruppiert. Im Diagramm sind jeweils zwei Klassen der jeweiligen Typen zu sehen.

Daneben ergänzt sie mit den Klassen ”CGJ3DClasses“ und ”CGJ3DTypes“ das Paket ”constants“. Erstere enthält für jede der Java3D-Klassen eine eindeutige ID als Konstante. Die zweite Klasse beinhaltet jeweils eine Konstante für jeden Typ, den ein Element besitzen kann, also ”GroupNode“, ”LeafNode“ oder ”NodeComponent“.

Die Controllerschicht wird um die Klasse ”J3DSourceCodeCreator“ erweitert, welche den Quellcode erstellt; die Daten dazu erhält sie von ”FrmMain“.

Um die einzelnen Elemente auch einfügen zu können, wird die Ebene der Oberfläche um die Klasse ”Java3DMenue“ ergänzt. Dieses Menü wird beim Start des Codegenerators in die Menüleiste eingefügt. Außerdem enthält das Paket mit den Klassen ”GroupNodePanel“, ”LeafNodePanel“ und ”NodeComponentPanel“ noch die Repräsentationen der jeweiligen Elementtypen für die Zeichenfläche. Sie erben alle von ”ElementPanel“.

4. Realisierung / Implementierung

4.1. Hilfsmittel

Zum Betrieb des Codegenerators werden aktuelle Versionen der JRE und des JDK benötigt. Aktuell ist jeweils die Version 6 Update 15. Diese sind auf folgenden Betriebssystemen lauffähig:

- Windows, alle Versionen ab 2000, jeweils in der x86- und x86-64 - Version (2000 nur x86)
- Linux, jeweils in der x86- und x86-64 - Version
- Solaris, jeweils in der x86-, x86-64- und SPARC - Version

Auf Apple Mac OS X übernimmt Apple die Verbreitung von Java, diese Versionen erscheinen meist etwas später. Prinzipiell gelten jedoch die selben Voraussetzungen.

Daneben wird eine Version der Java3D-Runtime benötigt, aufgrund der bereits erläuterten Probleme am Besten eine Version der Reihe 1.5.X.

Entwickelt wurde jeweils mit den aktuellen Versionen der oben genannten Programme. Als Entwicklungsumgebung kam NetBeans von Sun in Version 6.5.1, 6.7 und 6.7.1 auf Windows XP zum Einsatz.

4.2. Details

In den folgenden Abschnitten sollen bestimmte wichtige Funktionen einmal nochmals am Quellcode erläutert werden.

4.2.1. Daten

Daten eines Pfeiles Der Pfeil wird als Datenobjekt von der Klasse ”Arrow“ dargestellt und benötigt zum korrekten Erfüllen seiner Aufgaben die bereits oben erläuterten Attribute.

Listing 4.1: Attribute eines Pfeiles

```
/**
 * Typ des Pfeiles.
 */
private int typ = -1;
/**
 * Startpunkt des Pfeiles.
 */
private Point startPoint = null;
/**
 * ID des Startelements
 */
private int startID = -1;
/**
 * Endpunkt des Pfeiles.
 */
private Point endPoint = null;
/**
 * ID des Endelements
 */
private int endID = -1;
/**
 * Vektor mit Offsets der Zwischenpunkte zum Startpunkt.
 */
private Vector<Point> middlePointsOffset = new Vector<Point>();
/**
 * Vektor mit den absoluten Werten der Zwischenpunkte.
 */
private Vector<Point> middlePointsAbsolute = new Vector<Point>();
/**
 * Signalisiert, ob Autoroute aktiv ist oder ob der Pfeil manuell gerichtet wurde.
 */
private boolean autoroute = true;
/**
 * Farbe des Pfeiles.
 */
private Color currentColor = Color.BLACK;
```

Dabei stellen die dargestellten Werte die Standardwerte dar. Für die Speicherung der Zwischenpunkte wird ein Objekt der Klasse Vector verwendet.

Die Klasse "Arrow" stellt für alle Attribute "get"- und "set"-Methoden bereit. Zusätzlich dazu ermittelt sie , ob ein mit der Maus angeklickter Punkt auf der Zeichenfläche auf dem Pfeil oder auf einem Zwischenpunkt liegt. Die Methoden dafür sind "checkPointOnArrow(Point point)" und "checkPointOnMiddlePoint(Point point)".

Daten eines Elements Ein Datenelement wird durch eine Instanz einer von "SceneGraphElementData" erbenden Klasse dargestellt, die sich am Namen des Elements orientiert; zum Beispiel CGJ3DBranchGroup. Da die Attribute vererbt und von der erbenden Klasse modifiziert werden, sind sie durchgängig als "protected" deklariert.

Listing 4.2: Attribute eines Elements

```
/**
 * ID des Elements.
 */
protected int id;
/**
 * Klassenname des Elements als String
 */
protected String elementClassName = "";
/**
 * Klasse des Elements als ID (z.B. Box, TransformGroup, etc ....)
 */
protected int elementClassID = -1;
/**
 * Typ des Elements (Group, Leaf, NodeComponent)
 */
protected int type;
/**
 * Name des Elements
 */
protected String name = "";
/**
 * Die Farbe des Elements.
 */
protected Color color = Color.BLACK;
/**
 * Die Position des Elements auf dem Zeichen-Panel.
 */
protected Point location = null;
```

```
/**
 * Hoehe des Panels.
 */
protected int height = -1;
/**
 * Breite des Panels.
 */
protected int width = -1;
/**
 * Vector mit den Variablen dieses Elements
 */
protected Vector<SceneGraphElementAttributeData> data =
    new Vector<SceneGraphElementAttributeData>();
/**
 * Kinder des Elements.
 */
protected Vector<Integer> childNodes = new Vector<Integer>();
/**
 * Erlaubte Kindsknoten. Sind keine Einschraenkungen vorhanden, ist der Vektor leer.
 */
protected Vector<Integer> allowedChildren = new Vector<Integer>();
/**
 * Enthaelt die IDs der Knotenn, die von diesem Knoten referenziert werden, d.h.
 * wo dieser Knoten der Startknoten der Reference ist.
 */
protected Vector<Integer> referenceEndElements = new Vector<Integer>();
/**
 * Enthaelt die IDs der Elemente, die dieses Element referenzieren, d.h.
 * wo dieses Element Endelement der Referenz ist.
 */
protected Vector<Integer> referenceStartElements = new Vector<Integer>();
/**
 * Maximale Anzahl von Elternknoten. -1 steht dabei fuer unendlich
 * viele Elternknoten. Ist die 0 erreicht,
 * so kann keine Eltern-Kind-Beziehung mehr angelegt werden.
 */
protected int maxParents = -1;
/**
 * Elternknoten des Elements.
 */
protected Vector<Integer> parents = new Vector<Integer>();
/**
 * Vektor mit den erlaubten Typen, die bei einer Referenz Endelemente sein duerfen,
 * bei denen dieses Element das Startelement ist.
 */
protected Vector<ClassInfo> allowedReferences = new Vector<ClassInfo>();
/**
 * Vektor mit den Klasseninfomationen der Typen, die zum Erzeugen von
```

```
 * Quellcode unbedingt benoetigt werden.
 */
protected Vector<ClassInfo> neededReferences = new Vector<ClassInfo>();
/**
 * Signalisiert, ob bei einer Knotenkomponente erforderlich oder optional ist.
 */
protected boolean isNeededReference = false;
/**
 * Signalisert, ob Quellcode bereits erstellt wurde, damit Quellcode eines
  * Referenzobjektes nicht mehrfach erzeugt wird.
 */
protected boolean isSourceCodeAlreadyGenerated = false;
```

Als reines Datenobjekt stellt es Methoden zur Verfügung, um seine Daten zu ändern oder diese anderen Klassen bereit zu stellen. Auch hier wird wie beim Pfeil zum Speichern von den IDs ein Vektor verwendet.

Die erbenden Klassen können weitere Attribute deklarieren. In jedem Fall überschreiben sie die Methode ”GenerateCode(int tabs, FrmMain)“, welche den für das Datenobjekt spezifischen Quelltext erstellt.

Ein neues Datenobjekt wird durch Aufruf seines Konstruktors initialisiert, welcher als erste Anweisung die vererbte Methode ”init()“ aufruft.

Listing 4.3: Initialisierung eines Elements

```
public CGJ3DAmbientLight(int id, Color color)
{
    /* initialisieren - ID, Typ, Name, maximale Anzahl von Elternknoten */
    this.init(id, CGJ3DTypes.LEAF_NODE, CGJ3DClasses.AMBIENT_LIGHT, "AL", 1, color);
    ...
}
```

Diese bündelt alle wichtigen Eingaben und setzt sie auf den übergebenen Wert.

```
/**
 * Initialisiert das Objekt.
 * @param id    ID
 * @param typ   Typ des Elements
 * @param elementClassID Klassen-ID
 * @param elementClassName Name des Elements
 * @param maxParents    maximale Anzahl von Elternknoten
 * @param color Farbe des Knotens
```

```
 */
protected void init(int id, int typ, int elementClassID, String elementClassName,
    int maxParents, Color color)
{
    this.id = id;
    this.typ = typ;
    this.nodeClass = elementClassID;
    /* falls das Element ein Blattknoten oder eine Knotenkomponente ist, werden
     * die Vektoren fuer die Kinderknoten auf NULL gesetzt.
     */
    if(type == CGJ3DTypes.LEAF_NODE || type == CGJ3DTypes.NODE_COMPONENT)
    {
        childComponents = null;
        allowedChildren = null;
    }
    this.nodeName = elementClassName;
    this.maxParents = maxParents;
    this.color = color;
    name = "element" + id;
}
```

Danach werden weitere Angaben vorgenommen, die für die Semantik des Szenegraphen wichtig sind. Sie werden später erläutert.

Attribute eines Elements Jedes Attribut eines Elements wird durch eine Instanz der Klasse ”SceneGraphElementVariableData“ dargestellt und besitzt folgende Attribute:

Listing 4.4: Attribute eines Elementattributes

```
/**
 * Bezeichner fuer das Label vor dem Eingabefeld
 */
private String name;
/**
 * Variable fuer den Eingabewert - Object, muss auf Zielobjekt gecastet werden
 */
private Object returnValue;
/**
 * Konstante fuer ein GUI-Element, welches den Eingabewert enthaelt.
 */
private int guiElement = -1;
/**
 * Zusaetzliche Parameter fuer das GUI-Objekt.
 */
private Object guiParams = null;
```

```
/**
 * RegularExpression zum Pruefen, ob der eingegebene Wert einem korrekten Muster
 * entspricht. NULL, falls keine Pruefung erforderlich ist.
 */
private String regex = null;
```

Dabei erfolgt die Darstellung mit Hilfe der Klasse ”AttributeDataset“, welche die Daten für den Benutzer grafisch aufbereitet. Weiterhin ermöglicht sie mit der Methode ”getGUIElement()“ den Zugriff auf die Eingabekomponente, deren Inhalt mit der Methode im folgenden Listing auf Gültigkeit geprüft wird.

Beispielhaft für alle Datentypen wird hier die Prüfung vorgestellt, ob ein in ein Textfeld eingegebener String in den Zielwert geparst werden kann. Der reguläre Ausdruck wird bereits beim Anlegen der Variable im Konstruktor übergeben.

Listing 4.5: Prüfung des Eingabewertes eines Attributes auf syntaktische Korrektheit

```
public boolean checkReturnValue(JComponent component)
{
    switch(guiElement)
    {
        /* GUI-Element ist ein JTextField */
        case JTEXTFIELD:
            /* caste das uebergebene JComponent nach JTextField */
            JTextField temp = (JTextField) component;
            /* lese den Inhalt */
            String newValue = temp.getText();
            if(newValue.matches(regex) == true)
            {
                component.setBackground(Color.WHITE);
                return true;
            }
            else
            {
                component.setBackground(Color.RED);
                return false;
            }
        /* GUI-Element ist ein JButton */
        case JBUTTON:
            /* caste das uebergebene JComponent nach JButton */
            JButton tempButton = (JButton) component;
            /* Eingabevariable soll vom Typ Color3f sein */
            if(returnValue instanceof Color3f)
            {
```

```
                return true;
            }
            ...
    }
}
```

Bei weiteren verwendeten Typen von Eingabekomponenten ist das Vorgehen analog zum JButton. Da bei diesen allerdings keine freie Eingabe möglich ist, kann die Angabe eines regulären Ausdrucks bei Anlage des Attributs entfallen und wird mit NULL initialisiert.

Das andere Verfahren ist hier durch die in Ausschnitten wiedergegebene Methode ”setReturnValue(JComponent component)“ dargestellt und dient dem Setzen des Attributs auf den Wert, den der Benutzer in der Eingabekomponente eingetragen hat.

Es erfolgt dabei zuerst eine Unterscheidung nach der Art der Komponente. Dazu wird die Variable ausgewertet, die beim Anlegen des Attributs gesetzt wurde. Derzeit kommen die Komponenten JTextField, JComboBox, JButton und JCheckbox für diese Aktion zum Einsatz. Die übergebene JComponent in den jeweiligen Typ gecastet.

Listing 4.6: Ermittlung des Eingabewertes eines Attributes

```
/**
* Setzt den Eingabe- auf den uebergebenen Wert.
* @param component GUI-Komponente mit Eingabewert
*/
@SuppressWarnings("unchecked")
public void setReturnValue(JComponent component)
{
    switch(guiElement)
    {
        /* GUI-Element ist ein JTextField */
        case JTEXTFIELD:
            /* caste das uebergebene JComponent nach JTextField */
            JTextField tempTextField = (JTextField) component;
```

Nun erfolgt eine Unterscheidung nach dem Typ des Attributs. Dazu wird für jeden Typ geprüft, ob das Attribut eine Instanz von ihm ist. Im Falle einer Übereinstimmung wird je nach Art der Eingabekomponente deren Inhalt auf andere Weise ermittelt und das Attribut auf diesen Wert gesetzt.

Im Beispiel ist das Attribut vom Typ Point3f, der neue Wert liegt im Inhalt des dafür benutzten JTextField. Dieser muss dafür umgewandelt werden; im Beispiel sind die einzelnen Koordinaten jeweils durch ein Semikolon getrennt. Diese Strings werden nun in einen Float-Wert geparst und damit wird ein neuer Punkt erstellt, welchen das Attribut als Wert erhält.

```
            ...
            /* Eingabevariable ist vom Typ Point3f */
            else if(returnValue instanceof Point3f)
            {
                returnValue = ParsingUtilities.parsePoint3f(tempTextField.getText());
            }
            ...
            break;
        ...
        default:
            break;
    }
}
```

Die aufgerufene Methode zum Parsen wandelt den String in einen Point3f um. Sie ist Teil der Klasse ”ParsingUtilities“. Der String wird bereits vorher durch die Methode ”checkReturnValue(JComponent component) auf Gültigkeit geprüft.

```
/**
 * Erstellt einen Point3f aus einem String.
 * @param string    String
 * @return  Point3f
 */
public static Point3f parsePoint3f(String string)
{
    /* teile String in X, Y und Z */
    String[] coords = string.split(";");
    try
    {
        float x = Float.parseFloat(coords[0]);
        float y = Float.parseFloat(coords[1]);
        float z = Float.parseFloat(coords[2]);
        return new Point3f(x, y, z);
    }
    catch (NumberFormatException e)
    {
        System.out.println("Parsingfehler Point3f: " + e.getMessage());
```

```
        return null;
    }
}
```

Jedes als Attribut verwendeter Typ wird mit einer solchen Methode in einen String geparst. Eine Ausnahme bildet natürlich String als Typ selbst. Gegenteil ist die Speicherung eines Attributs in einem String. Dafür besitzt die Klasse "ParsingUtilities" für jeden Typ eine eigene Methode. Im Beispiel ist wieder die Methode für einen Point3f zu sehen.

```
/**
 * Erstellt einen String mit dem gegebenen Point3f.
 * @param value Point3f
 * @return  String, der den Point3f darstellt
 */
public static String point3fToString(Point3f value)
{
    return value.getX() + "f;" + value.getY() + "f;" + value.getZ() + "f";
}
```

Diese Umwandlung in einen String wird an zwei Stellen benötigt: Bei der Darstellung bestimmter Typen in einem Textfeld und zur Speicherung in einer XML-Datei. Deshalb werden diese Methoden gesammelt in der Klasse "ParsingUtilities" zur Verfügung gestellt.

Das gleiche Verfahren kommt auch in der Methode"getGUIElement()" zum Einsatz. Sie bestimmt den Typ des Attributs und generiert anhand obiger Angaben eine spezifische Eingabekomponente; für ein Objekt des Typs Point3f würde sie ein JTextField erstellen.

4.2.2. Semantik eines Szenegraphen

Mit dem in folgendem Listing gezeigten Code wird vor dem Anlegen einer Relation geprüft, ob diese überhaupt erstellt werden darf. Da es zwei Prüfungen gibt, ist hier die zweite angegeben, welche beim Klick auf ein Element ausgeführt wird. Die erste wird, wie bereits weiter oben erläutert, ausgeführt, wenn mit der Maus über das Element gefahren wird. Die Vorgehensweise ist dabei analog.

Im ersten Teil erfolgt die Prüfung, ob eine Vater-Kind-Beziehung angelegt werden darf. Dabei wird zuerst überprüft, welcher Teil der Relation hinzugefügt werden soll, der Start- oder der Endknoten. Dazu wird die Start-ID ausgewertet; ist sie "-1", so wird erst der Elternknoten eingefügt, welcher nur ein Gruppenknoten sein darf.

Listing 4.7: Prüfung auf semantische Korrektheit

```
if(main.getCreatePCRelation() == true)
{
    /* Startknoten ist noch nicht gesetzt */
    if(main.getCreateArrow().getStartID() == -1)
    {
        /* Parent-Knoten der Vater-Kind-Beziehung */
        SceneGraphElementData parent = main.getElementDataByID(
            main.getCreateArrow().getStartID());

        /* nur ein Gruppenknoten darf Kinder haben */
        if(this.getData().getType() == CGJ3DTypes.GROUP_NODE)
        {
            main.addStartOrEndToArrow(this);
            this.setCursor(main.getCursorPCRelationError());
        }
        else
        {
            this.setCursor(main.getCursorPCRelationError());
        }
    }
```

Ist die Start-ID ungleich "-1", so muss der Endknoten der Relation hinzugefügt werden. Aufgrund dessen, dass ein Startknoten nicht auch gleichzeitig der Endknoten sein kann, wird die Start-ID mit der des aktuellen Elementes verglichen. Nur wenn beide ungleich sind und das aktuelle Element nicht bereits einen Vater aufweist, wird es als Endknoten der Vater-Kind-Beziehung hinzugefügt. Da einige Knoten nur bestimmte Kinder besitzen dürfen, erfolgt anschließend noch eine Prüfung, ob der aktuelle Knoten ein Kindsknoten des Startknotens der Beziehung sein kann.

```
    /* Startknoten ist nicht dieser Knoten -> diese Komponente wird
    * Endknoten
    *
    * Endknoten kann hinzugefuegt werden, wenn die maximale Anzahl der
    * Elternknoten in dieser noch nicht erreicht ist.
```

```
    *
    * Cursor darf auf OK bleiben, weil
    * bereits der naechste Pfeil gezogen werden kann
    */
    else if(main.getCreateArrow().getStartID() != this.getID() &&
        (this.getData().getMaxParents() != 0)
        )
    {
        /* nur ein Gruppenknoten oder ein Blattknoten kann ein Kind sein */
        if(this.getData().getTyp() != CGJ3DTypes.NODE_COMPONENT  &&
            parent.isAllowedChild(this.getData().getElementClassID()))
        {
            main.addStartOrEndToArrow(this);
            this.setCursor(main.getCursorPCRelationOK());
        }
        else
        {
            this.setCursor(main.getCursorPCRelationError());
        }
    }
}
```

Bei einer Referenz ist das Vorgehen ähnlich dem bei einer Vater-Kind-Beziehung. Anders als bei dieser wird hier allerdings beim Anlegen des Startelements der Referenz geprüft, ob dieses überhaupt als Startelement einer Referenz in Frage kommt.

```
else if(main.getCreateReference() == true)
{
    /* Startelement der Referenz */
    SceneGraphElementData parent = main.getElementDataByID(
        main.getCreateArrow().getStartID());
    /* Startelement ist noch nicht gesetzt */
    if(main.getCreateArrow().getStartID() == -1 &&
        this.getData().canBeReferenceStartNode() == true)
    {
        main.addStartOrEndToArrow(this);
        this.setCursor(main.getCursorPCRelationError());
    }
```

Beim Anlegen des Endelements der Referenz wird hingegen ermittelt, ob das hinzuzufügende Element ein Endelement einer vom Startelement ausgehenden Referenz sein kann.

Nun werden, je nachdem ob der Nutzer dies in den Optionen eingestellt hat, noch

fehlende Referenzen gesucht. Werden solche gefunden, wird die Farbe des Labels mit dem Namen des Elements auf Rot gesetzt und in die Fehlervariable wird ein entsprechender Text geschrieben. Dieser gibt dem Benutzer genaue Auskunft hinsichtlich noch fehlender Referenzen. Die dazu erforderliche Prüfung übernimmt die Methode ”checkNeededReferences()“.

Werden keine fehlenden Referenzen gefunden, so wird die Schriftfarbe auf die Farbe des Elements gesetzt und die Fehlervariable gelöscht.

Je nach Vorlieben des Benutzers kann diese Prüfung bereits beim Erstellen von Elementen und Referenzen erfolgen oder erst beim Erstellen des Quellcodes. Erstere Einstellung ist vielleicht für Einsteiger sinnvoll, welche in der Regel noch wenig Erfahrung mit dem Aufbau eines Szenegraphen besitzen. Erfahrenere Benutzer werden zunächst häufig benutzte Elemente einfügen. Die dann angezeigten roten Markierungen könnten in diesem Fall als störend empfunden werden.

```
    /* Endelement ist noch nicht gesetzt */
    else if(main.getCreateArrow().getStartID() != this.getID() &&
        /* Start-ID ist nicht -1 */
        main.getCreateArrow().getStartID() != -1 &&
        /* Element darf Child des Startelements sein */
        parent.isAllowedParentReference(this.getData().getElementClassID(),
            FrmMain main))
    {
        main.addStartOrEndToArrow(this);

        /**
         * falls sofort auf fehlende Referenzen hingewiesen werden soll
         */
        if(main.getShowMissingReferencesImmediately() == true)
        {
            this.getData().checkNeededReferences(main);
            this.repaint();
        }
        /* falls nur beim Kompilieren auf fehlende Referenzen hingewiesen werden soll
         * fehlt allerdings bereits eine Referenz, wird geprueft, ob nach dem
         * Hinzufuegen dieser Referenz alle benoetigten vorhanden sind
         */
        else
        {
            if(this.getData().getErrorHelp().equals("") == false)
            {
```

```
                this.getData().checkNeededReferences(main);
                this.repaint();
            }
        }

        /* Cursor bleibt OK, weil bereits naechste Referenz angelegt werden kann */
        this.setCursor(main.getCursorPCRelationOK());
    }
}
```

Damit ein Element erkennt, welche Elemente Endelement einer von ihm ausgehenden Referenz sein dürfen, müssen die Klassen-IDs dieser Elemente beim Erzeugen des jeweiligen Elements angegeben werden. Dazu existiert der Vektor "allowedReferences". Dieser enthält Objekte vom Typ "ClassInfo". Diese wiederum beinhalten die Klassen-ID, den Klassennamen und die maximale Anzahl von Referenzobjekten dieses Typs, welche von dem aktuellen Element ausgehen dürfen. Die Methode "isAllowedParentReference(int id, FrmMain main)" überprüft beim Anlegen einer Referenz die angegebene ID. Eine Referenz wird angelegt, wenn ein "ClassInfo"-Objekt mit der gegebenen Klassen-ID vorhanden und noch nicht die maximale Anzahl von Referenzobjekten dieses Typs erreicht ist.

Diese statischen Angaben werden nicht gespeichert. Der Vorteil davon ist, dass diese Daten immer auf dem aktuellen Stand sind. Lädt der Benutzer eine alte Datei mit einer neuen Version des Programms, so kann er diese gleich weiter verwenden, da die Angaben aktuell sind.

```
/**
 * Fuegt Informationen zu Elementen hinzu, die bei einer Referenz Childs sein duerfen,
 * wobei dieses Element der Parent ist
 */
private void setAllowedReferences()
{
    this.allowedReferences.add(
        new ClassInfo(CGJ3DClasses.TRANSFORM_GROUP, "TransformGroup", 1));
    this.allowedReferences.add(
        new ClassInfo(CGJ3DClasses.BOUNDING_LEAF, "BoundingLeaf", 1));
}
```

Diese Maßnahmen stellen sicher, dass der Benutzer nur Relationen herstellen kann, die semantisch korrekt sind. Sie bieten jedoch keinen Schutz vor fehlenden Verbindun-

gen, denn wenn der Benutzer eine benötigte Referenz nicht erstellt, kann der Quellcode ebenfalls nicht kompiliert werden. Deswegen werden nach obigen erlaubten Referenz-IDs dem Element auch noch IDs mitgeteilt, ohne die eine Kompilierung nicht möglich ist. Fehlen diese beim Erstellen, so wird der Vorgang abgebrochen und der Benutzer durch einen rot markierten Namen des Elements darauf hingewiesen. Auch diese Angaben werden wie die erlaubten Referenzen nicht mit abgespeichert. Es gelten dafür die gleichen Gründe.

```
/**
 * Fuegt Informationen zu Elementen hinzu, die unbedingt referenziert werden
 * muessen, damit dieses Element korrekt funktioniert.
 */
private void setNeededReferences()
{
    this.neededReferences.add(new ClassInfo(CGJ3DClasses.ALPHA, "Alpha", 1));
    this.neededReferences.add(new ClassInfo(CGJ3DClasses.TRANSFORM3D,
        "Transform3D", 1));
    this.neededReferences.add(new ClassInfo(CGJ3DClasses.TRANSFORM_GROUP,
        "TransformGroup", 1));
    this.neededReferences.add(new ClassInfo(CGJ3DClasses.BOUNDING_LEAF,
        "BoundingLeaf", 1));
}
```

Dazu wird jeweils eine Instanz der ”ClassInfo“-Klasse erstellt, welche auf Anfrage durch die Methoden ”getClassName()“, ”getClassType()“ und ”getMaxReferenceObjects()“ sowohl den lesbaren Namen, die Klassen-ID und die maximale Anzahl von Referenzobjekten bereitstellt. Ersterer dient zum Anzeigen einer Hilfe für den Benutzer, letztere zum Prüfen durch das Programm.

4.2.3. Generierung von Quellcode

Die Erzeugung von Quellcode soll mit Hilfe der folgenden Listings näher erläutert werden. Dabei werden die verschiedenen Schritte genauestens ersichtlich.

Start der Erzeugung ist die Methode ”getCreateSceneGraph(int tabs)“ in der Klasse ”SourceCodeCreator“. Sie erstellt den Methodenrumpf der Methode ”createScreneGraph()“, welche wiederum den Szenegraph generiert. In dieser Methode wird die rekursive Erstellung des Quellcodes gestartet, indem die Methode ”generateCode()“ des

ersten Datenelementes aufgerufen wird. Dies ist immer eine BranchGroup.

Listing 4.8: Erzeugung von Quellcode

```
/**
* Gibt die Methode createSceneGraph als Quelltext zurueck.
* @param tabs  Anzahl der Tabs
* @return  Quelltext
*/
private String getCreateSceneGraph(int tabs)
{
    /* Tabs */
    String tab = getTabs(tabs);
    /* Daten der Komponenten */
    Vector<SceneGraphElementData> data = main.getSzeneGraphElementData();
    /* Ergebnis-String */
    String erg = "";
    erg =
        tab + "/**\n" +
        tab + " * Erstellt die BranchGroup.\n" +
        tab + " * @return\t" + data.firstElement().getName() + "\tBranchGroup\n" +
        tab + " */\n" +
        tab + "public BranchGroup createSceneGraph()\n" +
        tab + "{\n" +
        tab +   "\t//Erzeuge BranchGroup\n" +
        tab +  data.firstElement().generateCode(tabs + 1, main) + "\n" +
        tab +   "\treturn " + data.firstElement().getName() + ";\n" +
        tab + "}\n\n";

    return erg;
}
```

Die Methode ”generateCode()“ beginnt zunächst mit einer Prüfung, ob der Quellcode noch nicht erzeugt wurde. Falls dies der Fall ist, werden die benötigten Attribute aus dem Vektor ”data“ geladen. Anschließend muss die Angabe, ob der Quellcode bereits erstellt wurde, auf TRUE gesetzt werden, damit es wie oben erläutert nicht zu einer Endlosschleife kommt.

Danach wird das Element durch den Aufruf eines seiner Konstruktoren generiert. Dabei wird in der Regel der Konstruktor mit den meisten Parametern verwendet. Referenztypen werden im Allgemeinen hier nicht übergeben, sondern später per Methodenaufruf gesetzt, stattdessen werden hier NULL-Werte übergeben. In einigen wenigen, oben beschriebenen Ausnahmen erzeugt das allerdings eine NullPointerException. Das

betreffende Attribut muss in diesem Fall vor dem Konstruktor erstellt werden. Der Vorgang ist dabei analog zur Erstellung danach. Natürlich kann diese Variable beim Erzeugen der anderen Elemente nach der Konstruktorerstellung weggelassen werden.

```
/**
* Erstellt den Quellcode mit den gegebenen Parametern.
*/
@Override
public String generateCode(int tabs, FrmMain main)
{
    /* Tabs erstellen */
    String tab = SourceCodeCreator.getTabs(tabs);

    String code = "";

    if(isSourceCodeAlreadyGenerated == false)
    {
        float minimumAngle = (Float) this.data.get(0).getReturnValue();
        float maximumAngle = (Float) this.data.get(1).getReturnValue();

        isSourceCodeAlreadyGenerated = true;

        code =
            tab + "RotationInterpolator " + this.getName() + " =
                new RotationInterpolator(null, null);\n" +
            tab + this.getName() + ".setMinimumAngle(" + minimumAngle + "f);\n" +
            tab + this.getName() + ".setMaximumAngle(" + maximumAngle + "f);\n";
```

Falls das Element ein Gruppenknoten ist, werden nach dem Erzeugen des Konstruktors die einzelnen Kindsknoten eingefügt. Dabei werden alle Kindsknoten in einer Schleife durchlaufen und falls der Quellcode noch nicht erzeugt wurde, wird dies jetzt getan. Anschließend wird das Kind per ”addChild()“ in den Szenegraphen eingebunden. In dem gezeigtem Beispiel wurde dieser Teil des Quelltextes jedoch nur zur Erläuterung eingefügt. Im Quelltext des Programms ist dieser Teil nicht enthalten, weil ein RotationInterpolator als Blattknoten keine Kinder besitzen kann.

```
        /* Childs hinzufuegen */
        for(int i = 0; i < childNodes.size(); i++)
        {
            SceneGraphElementData temp =
                main.getNodeDataByID(childNodes.elementAt(i));
            /* Quellcode erzeugen */
```

```
            if(temp.isSourceCodeAlreadyGenerated() == false)
            {
                code += temp.generateCode(tabs + 1, main);
            }
            /* Quellcode einbinden */
            code += tab + this.getName() + ".addChild(" + temp.getName() + ");\n";
        }
```

Anschließend werden die Referenzen und deren Endelemente erstellt, bei welchen das aktuelle Element entsprechend das Startelement ist. Die hierbei erzeugten Knoten und Knotenkomponenten werden hier in den Quelltext eingebunden. Dies könnte zwar auch beim Erzeugen des Endelements getan werden, jedoch wie bereits erwähnt, soll damit eine gewisse Übersichtlichkeit gewahrt werden.

```
        for(int i = 0; i < referenceEndElements.size(); i++)
        {
            SceneGraphElementData temp =
                main.getElementDataByID(referenceEndElements.elementAt(i));
            /* Quellcode erstellen */
            if(temp.isSourceCodeAlreadyGenerated() == false)
            {
                code += temp.generateCode(tabs + 1, main);
            }
            /* Quellcode korrekt einbinden */
            switch(temp.getElementClassID())
            {
                case CGJ3DClasses.TRANSFORM_GROUP:
                    code += tab + this.getName() + ".setTarget(" + temp.getName()
                        + ");\n";
                    code += tab + temp.getName() +
                        ".setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);\n";
                    break;
                case CGJ3DClasses.BOUNDING_LEAF:
                    code += tab + this.getName() + ".setSchedulingBoundingLeaf(" +
                        temp.getName() + ");\n";
                    break;
                default:
                    break;
            }
        }
```

Sollte es sich bei dem Element zudem um ein Endelement einer Referenz handeln, so werden jetzt weitere Referenzen und deren Startelemente erstellt.

```
        for(int i = 0; i < referenceStartElements.size(); i++)
        {
            SceneGraphElementData temp =
            main.getElementByID(referenceStartNodes.elementAt(i));
            /* Quellcode erstellen -> eingebunden wird er vom Startelement der
             * Referenz !
             */
            if(temp.isSourceCodeAlreadyGenerated() == false)
            {
                code += temp.generateCode(tabs + 1, main);
            }
        }
    }
    return code;
}
```

4.3. Integration

Der Codegenerator ist unabhängig vom Betriebssystem; dass heißt, wenn für dieses eine JRE und ein JDK vorhanden sind, kann er problemlos ausgeführt werden.

Java bietet dafür vielfältige Möglichkeiten. Für Zeilenumbrüche und Separatoren können die entsprechenden Java Properties verwendet werden (“line.separator“ und ”file.separator“), die mit der Methode ”System.getProperty(String key)“ abgerufen werden können. Es ist erforderlich, dass diese durchgängig in der Programmierung Anwendung finden.

Eine weitere Besonderheit muss bei der Verwendung des Java-Compilers ”javac“ beachtet werden. Unter Windows besitzt diese Datei die Endung ”.exe“, unter Linux und UNIX dagegen keine Endung. Zwar bietet Java die Systemeigenschaft ”os.name“, welche mit obiger Methode ausgelesen werden kann. Sie liefert allerdings nur den Namen des Betriebssystems, zum Beispiel ”Windows XP“. Zur Prüfung, ob ”javac“ entsprechend mit oder ohne Endung verwendet werden muss, müssten mit dieser Lösung alle unterstützten Betriebssystem aufgelistet werden. Dies ist aufgrund der Vielzahl derer nicht vollständig möglich.

Also wird stattdessen im Quelltext zweimal die Prüfung durchgeführt, ob eine Datei

namens ”javac“ vorhanden ist. Zuerst wird das Vorhandensein für die Windows-Variante geprüft, anschließend für die Linux-Variante. Existiert keine der beiden Varianten, wird eine Fehlermeldung ausgegeben.

5. Zusammenfassung

5.1. Bewertung

Die derzeit vorliegende Version des Codegenerators besitzt alle am Anfang geforderten Merkmale. Er ermöglicht die Modellierung eines Szenegraphen und die Generierung von funktionsfähigem Quellcode. Dabei wurden Umsetzungen der wichtigsten Klassen der Java3D-API implementiert. Die Speicherung kann sowohl in XML als auch als serialisierte Objekte erfolgen. Des Weiteren ist das Exportieren in ein Bild vom Typ Portable Network Graphics möglich, über den ein Szenegraph auch ausgedruckt werden kann. Somit wurde das gesteckte Ziel erfüllt.

5.2. Kritik und Erfahrungen

Trotz umfangreicher Vorbereitungen mussten einige Teile der Anwendung mehrmals überarbeitet werden. Vor allem die Datenstrukturen waren davon betroffen. Bei der schrittweisen Implementierung der einzelnen Klassen von Java3D traten immer wieder Besonderheiten auf, die anfangs nicht absehbar waren. So war zum Beispiel anfangs der Vector ”allowedChildren“, welcher für einen Gruppenknoten erlaubte Kindsknoten enthält, nicht vorgesehen. Da allerdings von Anfang an bei den Referenzen ein solcher Vektor vorgesehen war, konnte diese Funktion schnell implementiert werden.

Zur Speicherung der Daten wurden anfangs serialisierte Objekte verwendet. Dies erwies sich als suboptimal, denn das Verfahren wies trotz seiner einfachen Implementierung mehrere Nachteile auf, welche sich vor allem beim Testen negativ bemerkbar

machten. Das wohl größte Defizit war dabei die Tatsache, dass aufgrund von Änderungen an Datenstrukturen während der Entwicklung alte Dateien nicht mehr lesbar waren. Es wäre hier sinnvoller gewesen, gleich die XML-Speicherung zu nutzen.

5.3. Mögliche Weiterentwicklung

Die derzeitige Version unterstützt keine Definition von eigenen Klassen, die auf Java3D-Klassen aufbauen, beziehungsweise von ihnen erben. Es ist also zum Beispiel nicht möglich, einen eigenen KeyBehavior einzufügen, der nur auf bestimmte Tastendrücke reagiert. Dies muss vom Benutzer nach der Erstellung des Quellcodes vorgenommen werden, setzt jedoch tiefer gehende Kenntnisse von Java3D voraus.

Daneben wären folgende weitere Erweiterungen möglich:

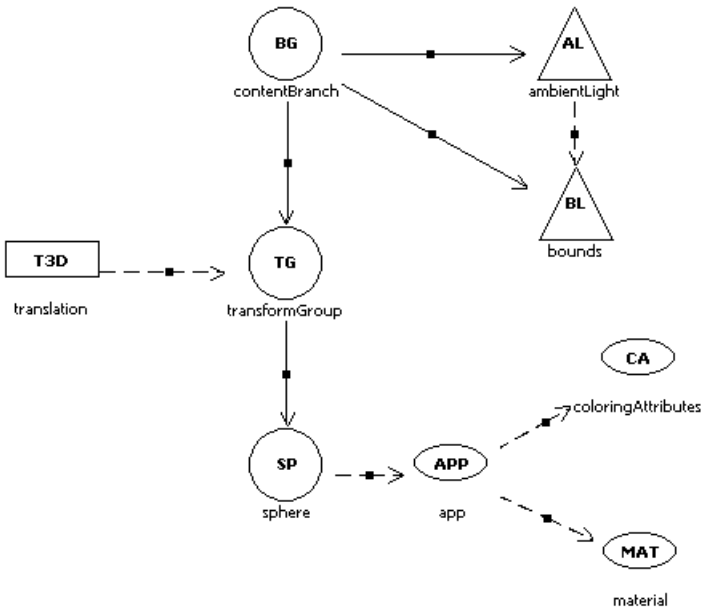
- Unterstützung von Copy & Paste
- Modellierung von Teilszenegraphen zur Schaffung von mehr Übersicht auf der Zeichenfläche
- Automatische Suche des JDK-Verzeichnisses
- Speicherung der Farbe für die nächste Komponente

Anhang

A. Entwicklungsstufen eines 3D-Programms

An dieser Stelle sollen noch einmal die einzelnen Ansichten eines Szenegraphen in all seinen Stadien gezeigt werden.

Ansicht im Codegenerator Die folgende Abbildung war bereits in der Einleitung zu sehen. Hier soll sie noch einmal aufgegriffen werden.



Ansicht als XML-Datei Das folgende Listing zeigt den gleichen Szenegraphen gespeichert in einer XML-Datei.

Listing A.1: Szenegraph als XML-Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<scenegraph>
    <sceneGraphElementsData>
        <sceneGraphElementData>
            <id>0</id>
            <elementName>BG</elementName>
            <elementClass>1001</elementClass>
            <type>1</type>
            <name>contentBranch</name>
            <color>0;0;0</color>
            <location>475;17</location>
            <height>60</height>
            <width>68</width>
            <data/>
            <childNodes>
                <childNode>1</childNode>
                <childNode>7</childNode>
                <childNode>8</childNode>
            </childNodes>
            <referenceEndElements/>
            <referenceStartElements/>
            <maxParents>1</maxParents>
            <parents/>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>1</id>
            <elementName>TG</elementName>
            <elementClass>1002</elementClass>
            <type>1</type>
            <name>transformGroup</name>
            <color>0;0;0</color>
            <location>470;154</location>
            <height>60</height>
            <width>77</width>
            <data/>
            <childNodes>
                <childNode>3</childNode>
            </childNodes>
            <referenceEndElements/>
            <referenceStartElements>
                <referenceStartElement>2</referenceStartElement>
            </referenceStartElements>
```

```
            <maxParents>1</maxParents>
            <parents>
                <parent>0</parent>
            </parents>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>2</id>
            <elementName>T3D</elementName>
            <elementClass>3004</elementClass>
            <type>3</type>
            <name>translation</name>
            <color>0;0;0</color>
            <location>332;153</location>
            <height>60</height>
            <width>60</width>
            <data>
                <sceneGraphElementAttributeData>
                    <name>Translation</name>
                    <typeOfReturnValue>Vector3f</typeOfReturnValue>
                    <returnValue>0.0f;0.0f;0.0f</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^(-{0,1}\d{1,}\.\d{1,}f;-{0,1}\d{1,}\.\d{1,}f;-{0,1}
                        \d{1,}\.\d{1,}f)$
                    </regex>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Neigung X</name>
                    <typeOfReturnValue>Integer</typeOfReturnValue>
                    <returnValue>0</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^-{0,1}\d{1,}$</regex>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Neigung Y</name>
                    <typeOfReturnValue>Integer</typeOfReturnValue>
                    <returnValue>0</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^-{0,1}\d{1,}$</regex>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Neigung Z</name>
                    <typeOfReturnValue>Integer</typeOfReturnValue>
                    <returnValue>0</returnValue>
                    <guiElement>1</guiElement>
```

```
                <guiParams>null</guiParams>
                <regex>^-{0,1}\d{1,}$</regex>
            </sceneGraphElementAttributeData>
        </data>
        <childNodes/>
        <referenceEndElements>
            <referenceEndElement>1</referenceEndElement>
        </referenceEndElements>
        <referenceStartElements/>
        <maxParents>-1</maxParents>
        <parents/>
        <isNeededReference>true</isNeededReference>
    </sceneGraphElementData>
    <sceneGraphElementData>
        <id>3</id>
        <elementName>SP</elementName>
        <elementClass>1006</elementClass>
        <type>1</type>
        <name>sphere</name>
        <color>0;0;0</color>
        <location>479;280</location>
        <height>60</height>
        <width>60</width>
        <data>
            <sceneGraphElementAttributeData>
                <name>Radius</name>
                <typeOfReturnValue>Float</typeOfReturnValue>
                <returnValue>0.4f</returnValue>
                <guiElement>1</guiElement>
                <guiParams>null</guiParams>
                <regex>^-{0,1}\d{1,}\.\d{1,}f$</regex>
            </sceneGraphElementAttributeData>
            <sceneGraphElementAttributeData>
                <name>Primflags</name>
                <typeOfReturnValue>int[]</typeOfReturnValue>
                <returnValue>0;</returnValue>
                <guiElement>5</guiElement>
                <guiParams>GENERATE_NORMALS,GENERATE_NORMALS_INWARD,
                    GENERATE_TEXTURE_COORDS,GENERATE_TEXTURE_COORDS_Y_UP,;
                    Sphere.GENERATE_NORMALS,Sphere.GENERATE_NORMALS_INWARD,
                    Sphere.GENERATE_TEXTURE_COORDS,
                    Sphere.GENERATE_TEXTURE_COORDS_Y_UP,
                </guiParams>
                <regex/>
            </sceneGraphElementAttributeData>
            <sceneGraphElementAttributeData>
                <name>Divisions</name>
                <typeOfReturnValue>Integer</typeOfReturnValue>
```

```
                    <returnValue>50</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^-{0,1}\d{1,}$</regex>
                </sceneGraphElementAttributeData>
            </data>
            <childNodes/>
            <referenceEndElements>
                <referenceEndElement>4</referenceEndElement>
            </referenceEndElements>
            <referenceStartElements/>
            <maxParents>0</maxParents>
            <parents>
                <parent>1</parent>
            </parents>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>4</id>
            <elementName>APP</elementName>
            <elementClass>3001</elementClass>
            <type>3</type>
            <name>app</name>
            <color>0;0;0</color>
            <location>582;280</location>
            <height>60</height>
            <width>60</width>
            <data/>
            <childNodes/>
            <referenceEndElements>
                <referenceEndElement>5</referenceEndElement>
                <referenceEndElement>6</referenceEndElement>
            </referenceEndElements>
            <referenceStartElements>
                <referenceStartElement>3</referenceStartElement>
            </referenceStartElements>
            <maxParents>-1</maxParents>
            <parents/>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>5</id>
            <elementName>CA</elementName>
            <elementClass>3002</elementClass>
            <type>3</type>
            <name>coloringAttributes</name>
            <color>0;0;0</color>
            <location>686;214</location>
```

```
            <height>60</height>
            <width>85</width>
            <data>
                <sceneGraphElementAttributeData>
                    <name>Farbe</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>1.0f;1.0f;1.0f</returnValue>
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Shade-Model</name>
                    <typeOfReturnValue>String</typeOfReturnValue>
                    <returnValue>ColoringAttributes.SHADE_GOURAUD</returnValue>
                    <guiElement>3</guiElement>
                    <guiParams>FASTEST,NICEST,SHADE_FLAT,SHADE_GOURAUD,;
                        ColoringAttributes.FASTEST,ColoringAttributes.NICEST,
                        ColoringAttributes.SHADE_FLAT,
                        ColoringAttributes.SHADE_GOURAUD,
                    </guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
            </data>
            <childNodes/>
            <referenceEndElements/>
            <referenceStartElements>
                <referenceStartElement>4</referenceStartElement>
            </referenceStartElements>
            <maxParents>-1</maxParents>
            <parents/>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>6</id>
            <elementName>MAT</elementName>
            <elementClass>3003</elementClass>
            <type>3</type>
            <name>material</name>
            <color>0;0;0</color>
            <location>700;335</location>
            <height>60</height>
            <width>60</width>
            <data>
                <sceneGraphElementAttributeData>
                    <name>AmbientColor</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>1.0f;1.0f;0.0f</returnValue>
```

```
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>EmissiveColor</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>0.0f;0.0f;0.0f</returnValue>
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>DifuseColor</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>1.0f;1.0f;1.0f</returnValue>
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>SpecularColor</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>1.0f;1.0f;1.0f</returnValue>
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Shininess</name>
                    <typeOfReturnValue>Float</typeOfReturnValue>
                    <returnValue>64.0f</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^-{0,1}\d{1,}\.\d{1,}f$</regex>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Color-Target</name>
                    <typeOfReturnValue>String</typeOfReturnValue>
                    <returnValue>Material.DIFFUSE</returnValue>
                    <guiElement>3</guiElement>
                    <guiParams>AMBIENT,AMBIENT_AND_DIFFUSE,DIFFUSE,EMISSIVE,SPECULAR,;
                        Material.AMBIENT,Material.AMBIENT_AND_DIFFUSE,Material.DIFFUSE,
                        Material.EMISSIVE,Material.SPECULAR,
                    </guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
            </data>
```

```
            <childNodes/>
            <referenceEndElements/>
            <referenceStartElements>
                <referenceStartElement>4</referenceStartElement>
            </referenceStartElements>
            <maxParents>-1</maxParents>
            <parents/>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
        <sceneGraphElementData>
            <id>7</id>
            <elementName>AL</elementName>
            <elementClass>2001</elementClass>
            <type>2</type>
            <name>ambientLight</name>
            <color>0;0;0</color>
            <location>658;17</location>
            <height>60</height>
            <width>60</width>
            <data>
                <sceneGraphElementAttributeData>
                    <name>LightOn</name>
                    <typeOfReturnValue>Boolean</typeOfReturnValue>
                    <returnValue>true</returnValue>
                    <guiElement>4</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Farbe</name>
                    <typeOfReturnValue>Color3f</typeOfReturnValue>
                    <returnValue>1.0f;1.0f;1.0f</returnValue>
                    <guiElement>2</guiElement>
                    <guiParams>null</guiParams>
                    <regex/>
                </sceneGraphElementAttributeData>
            </data>
            <childNodes/>
            <referenceEndElements>
                <referenceEndElement>8</referenceEndElement>
            </referenceEndElements>
            <referenceStartElements/>
            <maxParents>0</maxParents>
            <parents>
                <parent>0</parent>
            </parents>
            <isNeededReference>false</isNeededReference>
        </sceneGraphElementData>
```

```
        <sceneGraphElementData>
            <id>8</id>
            <elementName>BL</elementName>
            <elementClass>2008</elementClass>
            <type>2</type>
            <name>bounds</name>
            <color>0;0;0</color>
            <location>659;117</location>
            <height>60</height>
            <width>60</width>
            <data>
                <sceneGraphElementAttributeData>
                    <name>Mittelpunkt</name>
                    <typeOfReturnValue>Point3d</typeOfReturnValue>
                    <returnValue>0.0;0.0;0.0</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^(-{0,1}\d{1,}\.\d{1,};-{0,1}\d{1,}\.\d{1,};-{0,1}
                        \d{1,}\.\d{1,})$
                    </regex>
                </sceneGraphElementAttributeData>
                <sceneGraphElementAttributeData>
                    <name>Radius</name>
                    <typeOfReturnValue>Double</typeOfReturnValue>
                    <returnValue>1.0</returnValue>
                    <guiElement>1</guiElement>
                    <guiParams>null</guiParams>
                    <regex>^-{0,1}\d{1,}\.\d{1,}$</regex>
                </sceneGraphElementAttributeData>
            </data>
            <childNodes/>
            <referenceEndElements/>
            <referenceStartElements>
                <referenceStartElement>7</referenceStartElement>
            </referenceStartElements>
            <maxParents>1</maxParents>
            <parents>
                <parent>0</parent>
            </parents>
            <isNeededReference>false</isNeededReference>
            <boundingType>1</boundingType>
        </sceneGraphElementData>
    </sceneGraphElementsData>
    <arrows>
        <arrow>
            <type>1</type>
            <startID>0</startID>
            <startPoint>509;47</startPoint>
```

```
            <endID>1</endID>
            <endPoint>508;184</endPoint>
            <middlePointsRelative>
                <middlePointRelative>0;68</middlePointRelative>
            </middlePointsRelative>
            <middlePointsAbsolute>
                <middlePointAbsolute>509;115</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
        <arrow>
            <type>1</type>
            <startID>1</startID>
            <startPoint>508;184</startPoint>
            <endID>3</endID>
            <endPoint>509;310</endPoint>
            <middlePointsRelative>
                <middlePointRelative>0;63</middlePointRelative>
            </middlePointsRelative>
            <middlePointsAbsolute>
                <middlePointAbsolute>508;247</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
        <arrow>
            <type>1</type>
            <startID>0</startID>
            <startPoint>509;47</startPoint>
            <endID>7</endID>
            <endPoint>688;47</endPoint>
            <middlePointsRelative>
                <middlePointRelative>89;0</middlePointRelative>
            </middlePointsRelative>
            <middlePointsAbsolute>
                <middlePointAbsolute>598;47</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
        <arrow>
            <type>1</type>
            <startID>0</startID>
            <startPoint>509;47</startPoint>
            <endID>8</endID>
            <endPoint>689;147</endPoint>
            <middlePointsRelative>
```

```
            <middlePointRelative>90;50</middlePointRelative>
        </middlePointsRelative>
        <middlePointsAbsolute>
            <middlePointAbsolute>599;97</middlePointAbsolute>
        </middlePointsAbsolute>
        <autoroute>true</autoroute>
        <color>0;0;0</color>
    </arrow>
    <arrow>
        <type>2</type>
        <startID>7</startID>
        <startPoint>688;47</startPoint>
        <endID>8</endID>
        <endPoint>689;147</endPoint>
        <middlePointsRelative>
            <middlePointRelative>0;50</middlePointRelative>
        </middlePointsRelative>
        <middlePointsAbsolute>
            <middlePointAbsolute>688;97</middlePointAbsolute>
        </middlePointsAbsolute>
        <autoroute>true</autoroute>
        <color>0;0;0</color>
    </arrow>
    <arrow>
        <type>2</type>
        <startID>2</startID>
        <startPoint>362;183</startPoint>
        <endID>1</endID>
        <endPoint>508;184</endPoint>
        <middlePointsRelative>
            <middlePointRelative>73;0</middlePointRelative>
        </middlePointsRelative>
        <middlePointsAbsolute>
            <middlePointAbsolute>435;183</middlePointAbsolute>
        </middlePointsAbsolute>
        <autoroute>true</autoroute>
        <color>0;0;0</color>
    </arrow>
    <arrow>
        <type>2</type>
        <startID>3</startID>
        <startPoint>509;310</startPoint>
        <endID>4</endID>
        <endPoint>612;310</endPoint>
        <middlePointsRelative>
            <middlePointRelative>51;0</middlePointRelative>
        </middlePointsRelative>
        <middlePointsAbsolute>
```

```
                <middlePointAbsolute>560;310</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
        <arrow>
            <type>2</type>
            <startID>4</startID>
            <startPoint>612;310</startPoint>
            <endID>5</endID>
            <endPoint>728;244</endPoint>
            <middlePointsRelative>
                <middlePointRelative>58;-33</middlePointRelative>
            </middlePointsRelative>
            <middlePointsAbsolute>
                <middlePointAbsolute>670;277</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
        <arrow>
            <type>2</type>
            <startID>4</startID>
            <startPoint>612;310</startPoint>
            <endID>6</endID>
            <endPoint>730;365</endPoint>
            <middlePointsRelative>
                <middlePointRelative>59;27</middlePointRelative>
            </middlePointsRelative>
            <middlePointsAbsolute>
                <middlePointAbsolute>671;337</middlePointAbsolute>
            </middlePointsAbsolute>
            <autoroute>true</autoroute>
            <color>0;0;0</color>
        </arrow>
    </arrows>
    <dimension>831;412</dimension>
</scenegraph>
```

Ansicht als Java-Quellcode Folgendes Listing enthält den generierten Java-Quellcode.

Listing A.2: Quellcode eines Szenegraphen

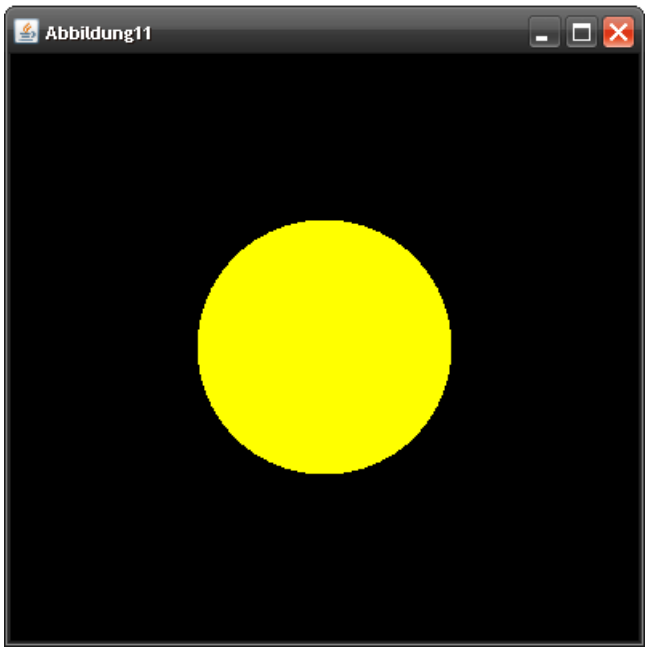
```
/**************************************/
/************ BEGINN IMPORTE ************/
/**************************************/
import javax.media.j3d.*;
```

```
import com.sun.j3d.audioengines.*;
import com.sun.j3d.audioengines.javasound.*;
import com.sun.j3d.exp.swing.*;
import com.sun.j3d.loaders.*;
import com.sun.j3d.loaders.lw3d.*;
import com.sun.j3d.loaders.objectfile.*;
import com.sun.j3d.utils.applet.*;
import com.sun.j3d.utils.audio.*;
import com.sun.j3d.utils.behaviors.interpolators.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.behaviors.sensor.*;
import com.sun.j3d.utils.behaviors.vp.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.geometry.compression.*;
import com.sun.j3d.utils.image.*;
import com.sun.j3d.utils.pickfast.*;
import com.sun.j3d.utils.pickfast.behaviors.*;
import com.sun.j3d.utils.scenegraph.io.*;
import com.sun.j3d.utils.scenegraph.transparency.*;
import com.sun.j3d.utils.shader.*;
import com.sun.j3d.utils.universe.*;
import javax.vecmath.*;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.*;
import java.awt.image.*;
import java.util.*;
import javax.imageio.*;
import java.net.URL;
/***************************************/
/************* ENDE IMPORTE *************/
/***************************************/

public class Abbildung11 extends Frame
{
    /**
     * Main-Methode
     * @param   args    Commandline-Argumente
     */
    public static void main(String[] args)
    {
        Abbildung11 program = new Abbildung11();
        program.setVisible(true);
        program.setSize(400, 400);
        program.setTitle("Abbildung11");
    };
```

```
/**
 * Konstruktor
 */
public Abbildung11()
{
    setLayout(new BorderLayout());
    Canvas3D myCan3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
    add("Center", myCan3D);
    BranchGroup scene = createSceneGraph();
    SimpleUniverse simpleU = new SimpleUniverse(myCan3D);
    simpleU.getViewingPlatform().setNominalViewingTransform();
    simpleU.addBranchGraph(scene);
    PhysicalEnvironment environment = simpleU.getViewer().getPhysicalEnvironment();
    AudioDevice device = new JavaSoundMixer(environment);
    device.initialize();
    environment.setAudioDevice(device);
    addWindowListener(new WindowAdapter()
    {
        @Override
        public void windowClosing(WindowEvent e)
        {
            dispose();
            System.exit(0);
        }
    });
}

/**
 * Erstellt die BranchGroup.
 * @return  contentBranch   BranchGroup
 */
public BranchGroup createSceneGraph()
{
    //Erzeuge BranchGroup
        BranchGroup contentBranch = new BranchGroup();
        TransformGroup transformGroup = new TransformGroup();
            Sphere sphere = new Sphere(0.4f, (Sphere.GENERATE_NORMALS), 50,
                                    new Appearance());
            sphere.setCapability(Sphere.ENABLE_APPEARANCE_MODIFY);
                Appearance app = new Appearance();
                    ColoringAttributes coloringAttributes =
                                                new ColoringAttributes();
                    coloringAttributes.setColor(new Color3f(1.0f, 1.0f, 1.0f));
                    coloringAttributes.setShadeModel(
                                            ColoringAttributes.SHADE_GOURAUD);
                app.setColoringAttributes(coloringAttributes);
                    Material material = new Material();
```

```
                        material.setAmbientColor(new Color3f(1.0f, 1.0f, 0.0f));
                        material.setEmissiveColor(new Color3f(0.0f, 0.0f, 0.0f));
                        material.setDiffuseColor(new Color3f(1.0f, 1.0f, 1.0f));
                        material.setSpecularColor(new Color3f(1.0f, 1.0f, 1.0f));
                        material.setShininess(64.0f);
                        material.setColorTarget(Material.DIFFUSE);
                    app.setMaterial(material);
                sphere.setAppearance(app);
            transformGroup.addChild(sphere);
                Transform3D translation = new Transform3D();
                translation.set(new Vector3f(0.0f, 0.0f, 0.0f));
                Transform3D translationrotX = new Transform3D();
                translationrotX.rotX(Math.toRadians(0));
                Transform3D translationrotY = new Transform3D();
                translationrotY.rotY(Math.toRadians(0));
                Transform3D translationrotZ = new Transform3D();
                translationrotZ.rotZ(Math.toRadians(0));
                translationrotX.mul(translationrotY);
                translationrotX.mul(translationrotZ);
                translation.mul(translationrotX);
                transformGroup.setTransform(translation);
        contentBranch.addChild(transformGroup);
            AmbientLight ambientLight =
                            new AmbientLight(true, new Color3f(1.0f, 1.0f, 1.0f));
                BoundingLeaf bounds = new BoundingLeaf(
                                    new BoundingSphere(
                                        new Point3d(0.0, 0.0, 0.0), 1.0));
            ambientLight.setInfluencingBoundingLeaf(bounds);
        contentBranch.addChild(ambientLight);
        contentBranch.addChild(bounds);
        contentBranch.compile();

        return contentBranch;
    }

}
```

Ansicht als ausgeführtes Java-Programm Die folgende Abbildung zeigt den Szenegraphen kompiliert und ausgeführt.



Literaturverzeichnis

Internetquellen

[1] Sun Microsystems, <www.sun.com>:
Java Systemvoraussetzungen
URL: <http://java.sun.com/javase/6/webnotes/install/system-configurations.html>,
Zugriff erfolgreich am 20.07.2009

[2] Java.net, <www.java.net>:
Java3D-Projekt
URL: <https://java3d.dev.java.net>,
Zugriff erfolgreich am 20.07.2009

[3] Sun Microsystems, <www.sun.com>:
Java3D-Guide
URL: <http://java.sun.com/javase/technologies/desktop/java3d/forDevelopers/J3D_1_3_API/j3dguide/index.html>,
Zugriff erfolgreich am 20.07.2009

[4] Java.net, <www.java.net>:
Java3D-API-Dokumentation
URL: <http://download.java.net/media/java3d/javadoc/1.5.2/index.html>,
Zugriff erfolgreich am 11.09.2009

Scripte

[5] Prof. Dr.-Ing. Rainer Gaudlitz,
Vorlesung ”Grafiksysteme“
Hochschule Mittweida (FH), 2008

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bearbeitungsort, Datum	Unterschrift