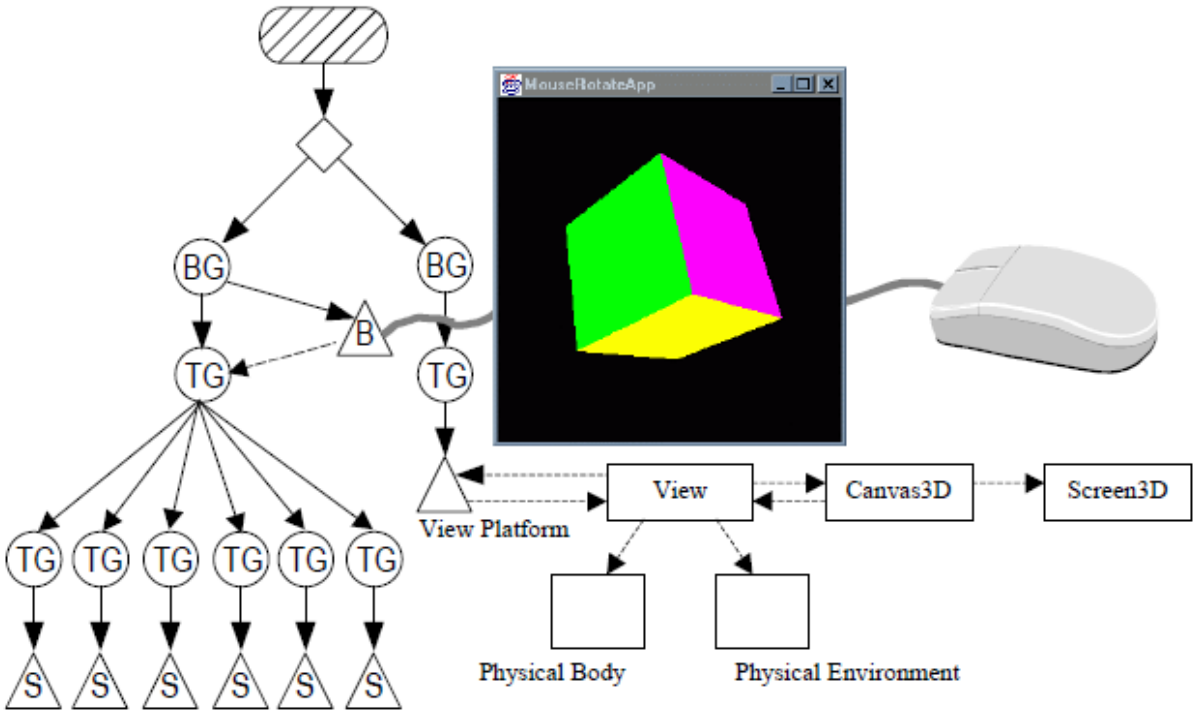
Java 3D
Interaktion



Eine Ausarbeitung von
Manuel Schmidberger
im Rahmen der Lehrveranstaltung
Graphische Interaktive Systeme

Hochschule Fulda WS 2010/2011

Inhaltsverzeichnis

1. Einleitung

Die Programmiersprache Java erfreut sich immer größerer Beliebtheit. Dank der Objektorientierung und der Plattformunabhängigkeit sowie der großen Anzahl an vielseitigen Erweiterungen ist sie für beinahe jede Problemstellung geeignet. Neben den komfortablen Entwicklungswerkzeuge wie Eclipse oder die Netbeans IDE, sind auch die umfangreichen Bibliotheken und eine gut übersichtliche Dokumentation dieser Sprache ein großer Grund, dass Java immer interessanter für Entwickler wird. Mit der Installation eines einfachen Plug-in ist es möglich in Java auch dreidimensionale Objekte zu modellieren, rendern, sowie das Verhalten und die Ansicht zu steuern. Das sogenannte Java 3D ist eine Klassenbibliothek von Java-Klassen zur Erzeugung, Manipulation und Darstellung dreidimensionaler Grafiken innerhalb Java-Anwendungsprogrammen und -Applets. Java 3D ist seit Sommer 2004 als Open Source freigegeben.
Java 3D kapselt die Funktionalität der zugrundeliegenden OpenGL- oder DirectX-Schnittstelle in ein leichter verständliches objektorientiertes Programmkonzept auf Basis eines Szenengraphen. Im Szenengraph wird der logische Aufbau der darzustellenden Objekte auf eine gleichartig aufgebaute, baumähnliche Struktur abgebildet, die im Wesentlichen aus Definitionen von Transformationen und Geometriedaten besteht. Die so strukturierte Sicht der Szene erlaubt eine komfortable Handhabung der Objekte.

2. Installation

2.1 Voraussetzungen

Um Programme mit Java 3D entwickeln zu können werden folgende Installationen benötigt:

- Java SDK
- Java 3D

Es empfiehlt sich für die Entwicklung ein geeignetes Programm zu benutzen. Ich persönlich empfehle hierzu die Entwicklungsumgebung Eclipse, da sie kostenlos und sehr umfangreich ist. Mit Eclipse werden folgende Speicherplatzanforderungen für Java 3D benötigt:

- 100 MB Eclipse
- 300 MB Java SDK
- 10 MB Java 3D

2.2 Java SDK installieren

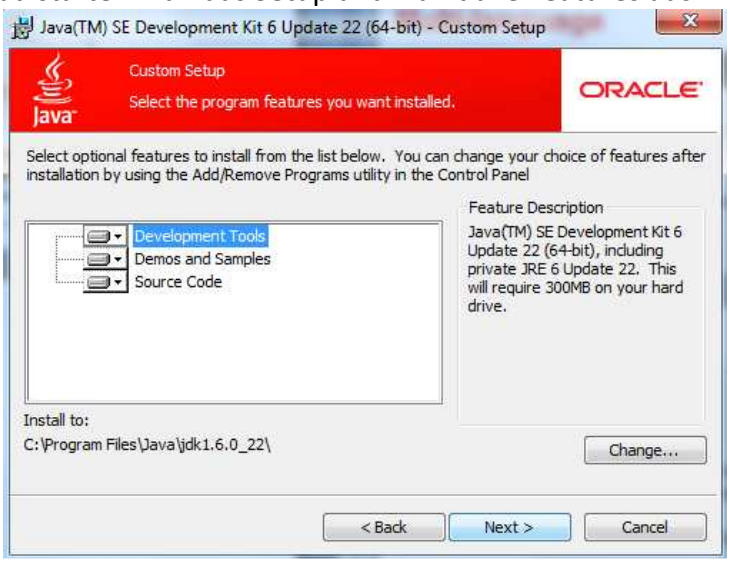
Die benötigte Installationsdatei findet man unter:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Dort wählt man "Java" zum downloaden aus.

Nun muss die richtige Datei für das Betriebssystem und den Prozessor (32Bit oder 64Bit) ausgewählt werden.

File Description and Name	Size
Java SE Development Kit 6u22 jdk-6u22-windows-x64.exe	66.40 MB

Nach dem Download starten man das Setup und wählt alle Features aus.



2.3 Java 3D installieren

Die benötigte Installationsdatei findet man unter:
https://java3d.dev.java.net/#Downloads

Hier klickt man unter Downloads auf die Java 3D "binary downloads".

Downloads

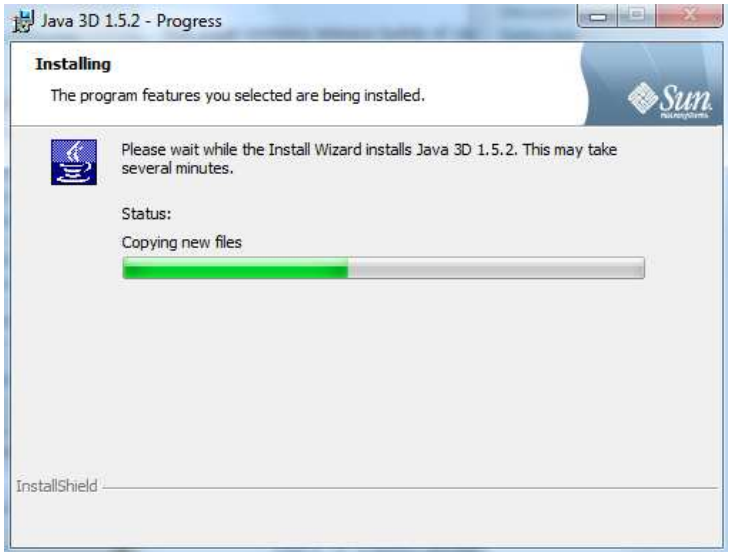
Java 3D binary downloads are available on java.net. Three types of builds are provided: Release (or FCS) builds; Stable (experimental, early access) builds; and Daily builds.

Hier kann man manuell die Binery's downloaden oder direkt die Installationsdatei. Nun wählt man die Installationsdatei für das Betriebssystem und den Prozessor (32bit oder 64bit) aus.

Installers

LICENSE
THIRD-PARTY-LICENSE-README
j3d-1_5_2-linux-amd64.bin
j3d-1_5_2-linux-i586.bin
j3d-1_5_2-solaris-sparc.bin
j3d-1_5_2-solaris-x86.bin
j3d-1_5_2-windows-amd64.exe
j3d-1_5_2-windows-i586.exe

Setup starten und nun Java 3d installieren.

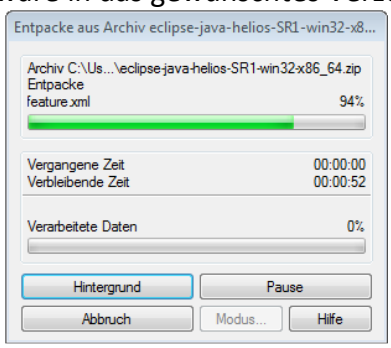


2.4 Eclipse installieren

Die benötigte Installationsdatei findet man unter:
http://www.eclipse.org/downloads/

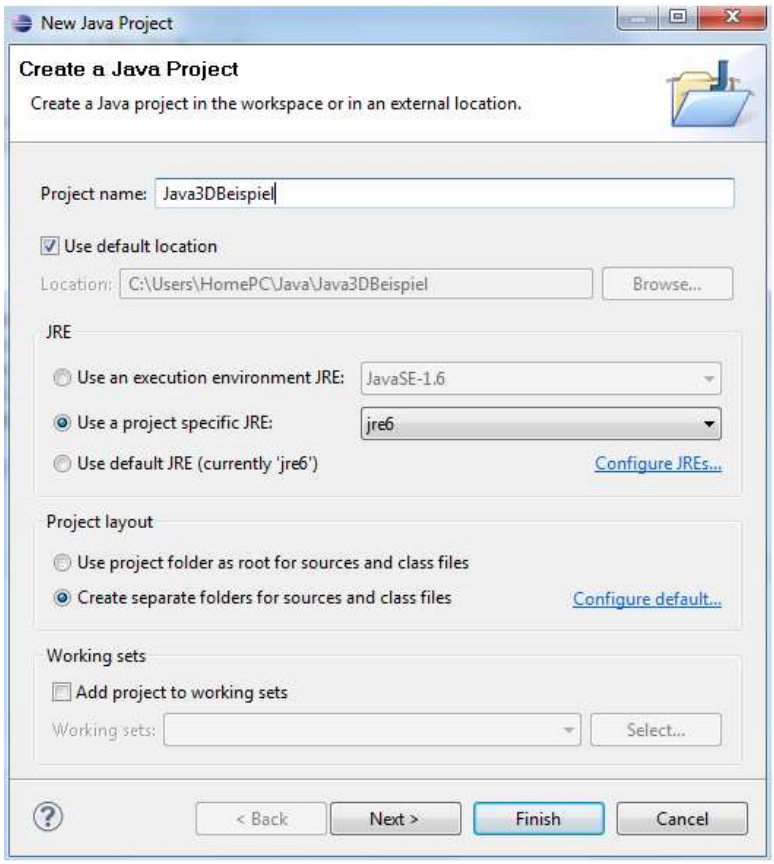
Für Java 3D reicht die normal Java Eclipse IDE Entwicklungsumgebung. Downloaden Sie die gewünschte Software für ihr System.

Zuletzt entpacken man die Software in das gewünschtes Verzeichnis.

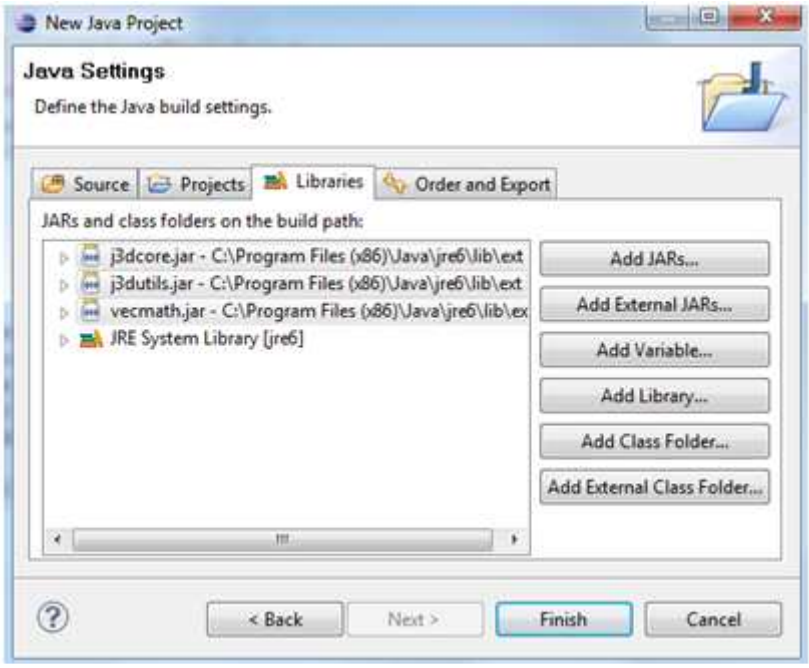


2.5 Erstes Testprogramm mit Java 3D

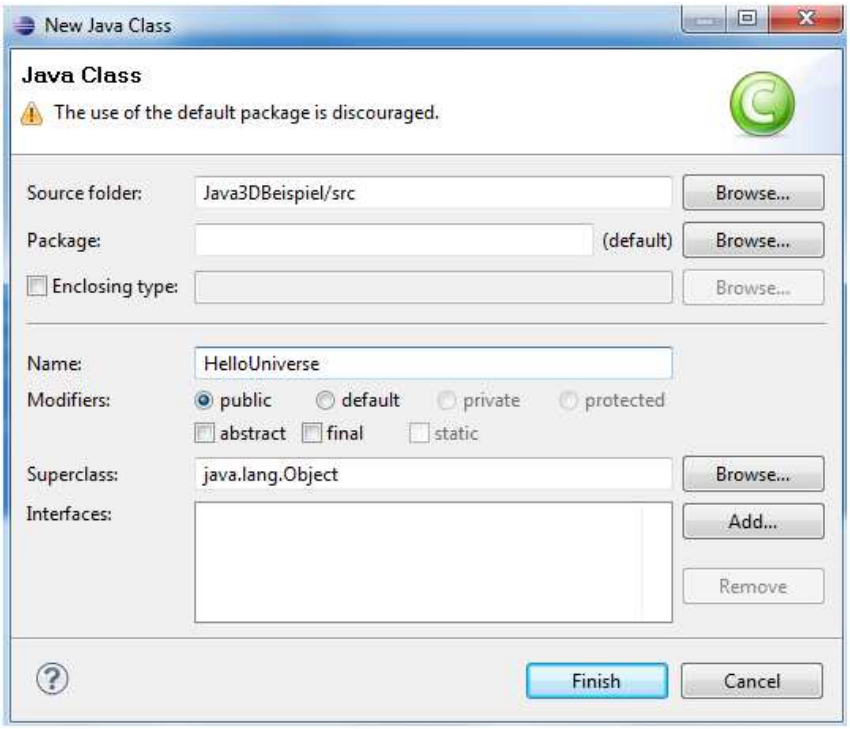
Dazu startet man Eclipse und erstellt ein neues Java-Projekt. Es müssen folgenden Einstellungen vorgenommen werden. (File -> New -> Java-Project)



Anschließend klickt man auf "Next" und dann auf "Libraries".
Hier muss nun die Java 3D Bibliothek-Dateien eingebunden werden in dem man auf "Add External JARs" klickt und die 3 Java Dateien auswählt (j3dcore.jar, j3dutils.jar, vecmath.jar). Diese sollte unter "C:\Programme\Java\bin\ext" zu finden sein, sofern Java in das Standardverzeichnis installiert wurde. Die Settings sollten dann wie folgt aussehen. Weiter mit klick auf Finish.



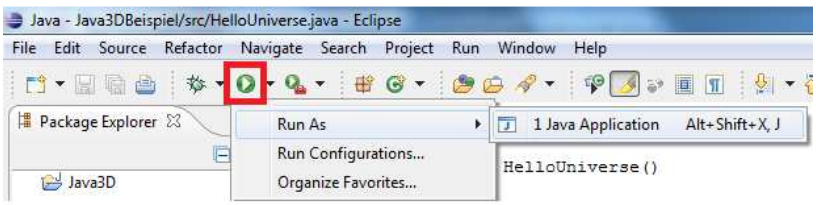
Nun erstellt man eine neue Java Klasse. (Rechtsklick auf ihr Projekt -> New -> Class)



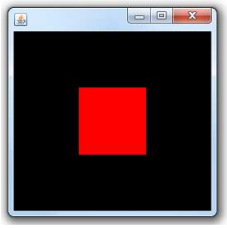
Zuletzt noch den folgenden Programm-Code kopieren und in die Klasse einfügen.

```
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.geometry.ColorCube;
import javax.media.j3d.BranchGroup;

public class HelloUniverse
{
	public HelloUniverse()
	{
		SimpleUniverse universe = new SimpleUniverse();
		BranchGroup group = new BranchGroup();
		group.addChild(new ColorCube(0.3));

	universe.getViewingPlatform().setNominalViewingTransform();
	{
		universe.addBranchGraph(group);
	}
	public static void main (String[] args)
	{
		new HelloUniverse();
	}
}
```

Das Programm Compilern mit "Alt + Shift + X, J" oder mit dem "Run -> Run As -> Java Application".



Wenn nun alles richtig gemacht wurde, sollte folgendes Bild erscheinen.



Nun ist die Entwicklungsumgebung richtig eingerichtet und es kann nun das eigene Java 3D Programm entwickelt werden.

3. Neuerungen in Java 3D

Aktuelle Java 3D Version 1.5.2

Erweiterungen

- Quellcode-Lizenz geändert zu GPL v2 + CLASSPATH

Behobene Fehler

Issue	Description
179	OrbitBehavior.setRotationCenter gives spurious view shift
483	NullPointerException when writing Font3D using scenegraph.io
506	NullPointerException: Calling getNominalSensorRotation in WandViewBehavior when internal nominalSensorRotation is null causes a NullPointerException.
510	J3DGraphics2D context lost when canvas changes frame
513	Down-rev D3D driver can cause JVM to crash
514	NPE in Wonderland : triggered in cached bounds computation
519	IntersectionInfo.getGeometry return null for IndexedArrayGeometry
525	JOALMixer only playing one sample
532	Background geometry BG isn't saved with SceneGraphFileWriter
534	ClassNotFoundException when running applet if Java 3D installed into JRE
538	Machine precision bug in AxisAngle4d and Quat4d
540	ArrayIndexOutOfBoundsException when calling setPickable
541	Bound.closest_point() method creates unused Matrix3d
543	J3DClock does not adjust to clock skew
544	GroupRetained.getBounds() should return BoundingBox?
545	Update docs to discourage installing Java 3D into JRE
548	RFE - Disable getLock() / unLock() on non-alive GeometryArray
555	Muting a PointSound causes a ClassCastException
560	Use D3DCREATE_FPU_PRESERVE flag on D3D pipeline
561	Decrease memory footprint of BoundingBox
562	Error occurs when Canvas3D removed from View
563	Background cloneNode() fails with Background geometry
567	Update license to GPL v2 + CLASSPATH
569	ImageComponent.ALLOW_IMAGE_READ is false
583	A disposed Graphics2D remains in Canvas3D after removal and addtion
585	ClassCastException in TransformStructure.java

JOGL Rendering Pipeline

- Die JOGL Rendering-Pipeline wird nun auf allen Plattformen unterstützt.

4. Verhalten (Behavior)

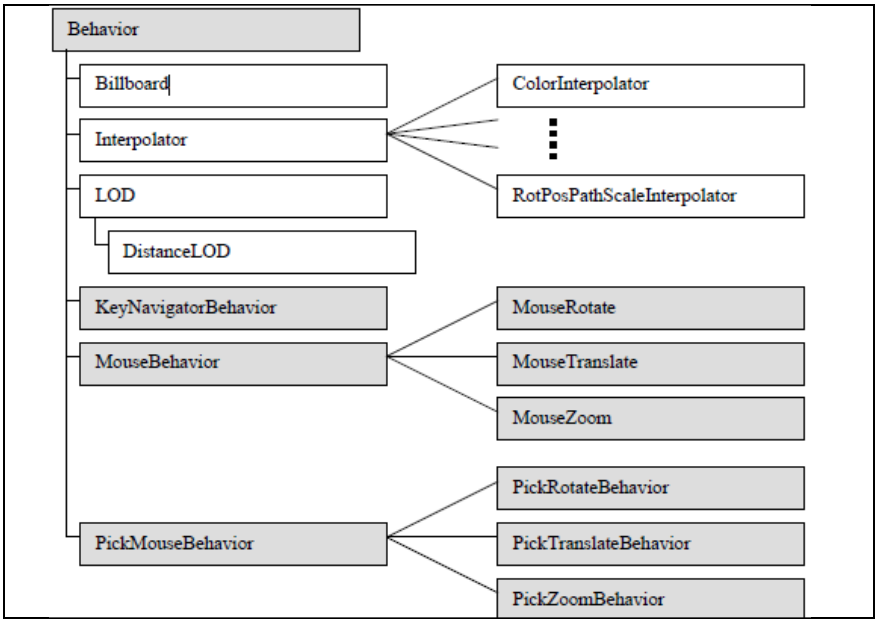
Da ein Verhalten eine Verbindung zwischen einem Stimulus und einer Aktion ist, unter Berücksichtigung aller möglichen Kombinationen von Reizen (stimuli) und möglichen Aktionen, müssen viele Anwendungen der Verhaltensobjekte betrachtet werden. Die folgende Tabelle zeigt den Umfang der Möglichkeiten eines Verhaltens. Die Tabelle listet nicht alle möglichen Aktionen auf, sondern beschränkt sich auf die wichtigsten. In der linken Spalte sind die möglichen Reize gelistet. Die rechten Spalten listen die möglichen Veränderungen auf.

	Das zu veränderte Objekt			
Stimulus (Grund für die Änderung)	**Verschiebungs-gruppe** (Visuelles Objekt ändert die Orientierung oder den Standort)	**Geometrie** (Visuelles Objekt ändert die Fläche oder Farbe)	**Szenen-Graph** (Objekte hinzufügen, entfernen oder verändern)	**View** (Ändern des Standortes oder die Richtung des Betrachter)
Anwender	Interaktion	Anwendungsspezifisch	Anwendungsspezifisch	Navigation
Kollision	Visuelles Objekt ändert die Orientierung oder den Standort	Visuelle Objekte verändern das Erscheinungsbild bei einem Zusammenstoß	Visuelle Objekte verschwindet in Kollision	Ansicht ändert nach Kollision
Zeit	Animation	Animation	Animation	Animation
Lokalisierung	Reklametafel	Detaillevel (LOD)	Anwendungsspezifisch	Anwendungsspezifisch

Wenn ein Benutzer sich durch ein Programm navigiert verändern sich die Objekte meist per Maus klick oder Tastatureingabe. Die Bewegung der Darstellung ist eine Interaktion, da es ein direktes Ereignis auf die Aktion des Users ist. Veränderung als Folge der Ansicht, sind indirekt durch den Benutzer verursacht und sind daher Animationen.

4.1 Verhaltens Basics

Es ist wichtig, die Funktionsweise und Programmierüberlegungen von der Verhaltens Klasse zu verstehen. In diesem Abschnitt erkläre ich die Verhaltens Klasse und gebe einen Ablaufplan für die Programmierung einer benutzerspezifischen Verhaltens Klasse. Anschließend noch ein einfaches Anwendungsbeispiel, das eine Verhaltensklasse benutzt.



Spezialisierung der Behavior im Java 3D API Kern und die Utility-Pakete

4.2 Eine Verhaltensklasse schreiben

Um eine eigene Behavior Klasse zu erstellen muss die Initialisierungs und processStimulus Methoden der abstrakten Verhaltens Klasse implementiert werden. Somit hat die eigenen Behavior Klasse mindestens einen Konstruktur und auch weitere Methoden.
Bei den meisten Verhalten wird dieses mit einen Szenengraph-Objekt gemacht, um das Verhalten zu beeinflussen. Die Verhaltensklasse benötigt eine Referenz auf das Objekt das geändert werden soll. Der Konstruktor kann dazu benutzt werden, um die Referenz auf das zu veränderte Objekt zu setzen. Dies kann durch den Konstruktor realisiert werden. Wenn dies nicht im Konstruktor definiert wird, muss die Referenz in einer Methode realisiert werden.
Wenn die Initialisierungs-Methode aufgerufen wird, wird der Szenengraph welche die Verhaltensklasse enthält zum Leben erweckt. Die Initialisierungs-Methode ist verantwortlich für die Festlegung des Trigger-Ereignis und die Einstellung des Ausgangszustands von den Zustandsgrößen. Der Trigger ist als WakeUpCondition Objekt, oder eine Kombination von WakeUpCondition Objekten spezifiziert. Die processStimulus Methode wird aufgerufen, wenn das angegebene auszulösende Ereignis für das Verhalten auftritt. Die Methode ist auch verantwortlich für die Reaktion auf das Ereignis.
Wie bereits gesagt, wird für die Verhaltensklasse nur der Konstruktor, die Initialisierungsmethode und die processStimulus Methode benötigt. Der Konstruktor um die Referenz auf das zu veränderte Objekt zu setzen. Die Initialisierungsmethode um den Auslöser zu setzen. Und die processStimulus Methode um die Auslöskriterien zu bestimmen.

1. Konstruktur schreiben
   eine Referenz auf das Objekt setzten, das geändert werden soll
2. Initialisierung
   Überschreiben der Initialisierungsmethode mit dem eigenen Wake-up Kriterium
3. processStimulus
   Überschreiben sie die processStimulus Methode mit ihren Auslöskriterien

Dies sind die grundlegenden Schritte zum Erstellen einer benutzerdefinierten Verhaltensklasse. Komplexe Verhaltensweise erfordern mehr als nur die oben genannten Programmierungen.

4.3 Benutzerspezifische Verhaltensklasse (SimpleBehavior)

Um eine Verhaltensklasse zu erzeugen benötigen sie nur die Referenz auf eine TransformGroup (Das Objekt das geändert werden soll) und eine Winkelvariable. Als Reaktion auf einen Tastendruck wird der Winkel der Variable geändert und der neue Winkel wird auf die TransformGroup gesetzt. Da es nur eine mögliche Trigger WakeUp Bedingung gibt, dekodiert die processStimulus Methode nicht die Trigger WakeUp Bedingung. Es ist möglich weitere Tasten durch dekodieren zu bestimmen, welche Taste oder welche Tastenkombination gedrückt wurde.
Die processStimulus Methode erhöht immer die Winkelvariable und nutzt diese dann um das zu veränderte TransformGroup Objekt anzupassen. Die letzte Aufgabe der processStimulus Methode ist es, den Trigger zu resetten.

1. Erzeuge SimpleBehavior und zum Verändern setze das TransformGroup Objekt

```
SimpleBehavior(TransformGroup targetTG) {
        this.targetTG = targetTG;
}
```

2. Initialisierung des Verhaltens und setze die WakeUp Bedingungen

```
public void initialize() {
        this.wakeupOn(new WakeupAWTEvent(KeyEvent.KEY_PRESSED));
}
```

3. Wird ausgeführt, wenn in Java 3D der gegebene Reiz auftritt

```
public void processStimulus(Enumeratin criteria) {
        angle += 0.1;
        rotation.rotY(angle);
        targetTG.setTransform(rotation);
        this.wakeupOn(new WakeupAWTEvent(KeyEvent.KEY_PRESSED));
}
```

SimpleBehaviorApp: *Den Gesamten Source-Code finden sie im Anhang.*

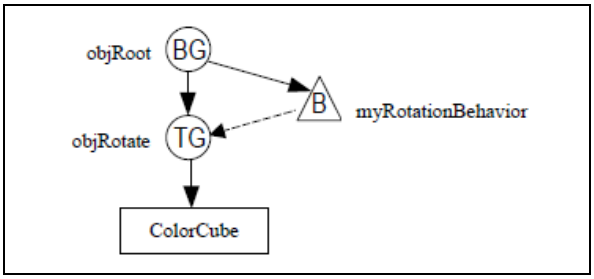
In diesen drei Schritten können bereits die ersten Programmierfehler auftreten. Meist wird Vergessen den Verhaltens-Trigger zu setzen und zu resetten. Selbstverständlich gilt, wenn der zu initialisierende Trigger nicht in der Initialisierung Methode gesetzt ist, kann das Verhalten nie geltend gemacht werden. Etwas weniger offensichtlich ist, dass der Trigger in der processStimulus Methode wieder gesetzt werden muss, wenn eine Wiederholung des Verhaltens gewünscht ist. Viele Verhalten benötigen nur ein TransformGroup Objekt.

4.4 Eine Verhaltensklasse benutzen

Der erste Schritt um eine Verhaltensklasse einzubinden besteht darin, sicherzustellen dass der Szenengraph die Bestimmung für das Verhalten enthält. Z.B. um die SimpleBehavior aus dem vorigen Abschnitt zu benutzen, muss die TransformGroup im Szenengraphen über dem Objekt rotiert werden. Für die Unterstützung des Verhaltens muss eine Instanz von der Klasse zum Szenengraph hinzugefügt werden. Wenn diese nicht ein Teil vom Live-Szenengraph ist, gibt es keine Möglichkeit das Verhalten zu initialisieren. Sollte ein Verhaltensobjekt nicht ein Teil eines Szenengraphen sein, hat dieses keinen Nutzen und wird beseitigt. Der letzte Schritt um ein Verhalten hinzuzufügen besteht darin die Scheduling Grenzen bereitzustellen. Zur Verbesserung der Effizenz benutzt Java 3D die Scheduling Grenzen um die culling Ausführung zu verbessern. Das Verhalten ist nur aktiv, wenn die Scheduling Grenzen ddie ViewPlatform's Aktivierung schneidet. Nur aktive Verhalten sind berechtigt Reize zu empfangen. Auf diese Weise können Impulse für einige Verhaltensweisen ignoriert werden. Der Programmierer hat die Kontrolle über die culling Ausführung durch die Auswahl der Scheduling Grenzen für das Verhalten.

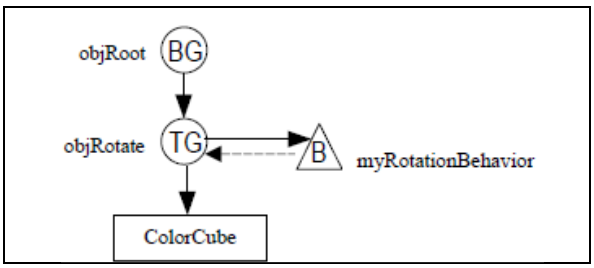
1. Den Szenengraphen festlegen
2. Verhaltensobjekt in den Szenengraphen einfügen und eine Referenz auf das zu veränderte Objekt setzen
3. Festlegen der Scheduling Grenzen
4. Setze Schreib- und Lesefähigkeiten für das Zielobjekt

Die Szenengraph Abbildung für den Inhalts-Teilbaumgraph der SimpleBehaviorApp Klasse.



Szenengraph Abbildung

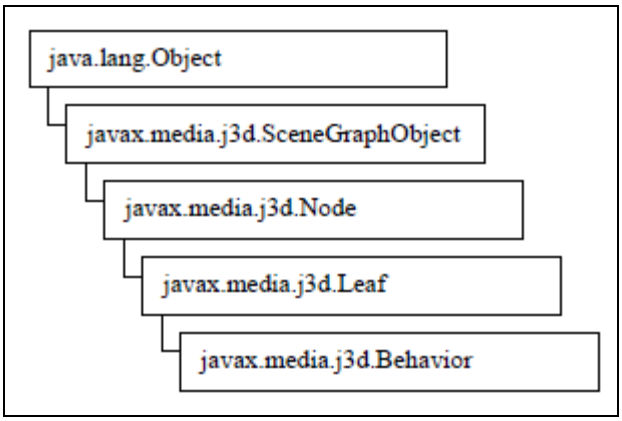
Das Diagramm zeigt deutlich die Beziehung zwischen dem Verhaltens Objekt und dem zu veränderten Objekt, das TransformGroup Objekt. Das Beispiel dreht einen ColorCube, jedoch ist die Verhaltensklasse nicht nur auf das drehen beschränkt. Es kann jedes visuelle Objekt oder einen Teil des Szenengraphen, dar ein Kind des TransformGroup Objekt ist, drehen. Dieses Beispiel soll nicht alle Möglichkeiten von Verhaltensklassen demonstrieren, sondern nur als Ausgangspunkt für das Verhalten dienen.
Ein Verhalten kann überall im Szenengraph platziert werden. Die Probleme beim Picking in einem Szenengraph für eine Verhaltens Objekt sind die Auswirkungen auf die Scheduling Grenzen und die Code Wartung. Im erstellten Szenengraph im SimpleBehaviorApp, ist das SimpleBehavior Objekt und der ColorCube nicht Gegenstand eines lokalen Koordinatensystem. In dieser Beispiel-Anwendung führt das zu keinem Problem. Das TransformGroup Objekt, aus dem Beispiel, rotiert nur den ColorCube. Die Scheduling Grenzen für das myRotationBehavior Objekt umschließt das ColorCube Objekt. Somit ist es möglich mit dem ColorCube zu interagieren, sobald er sichtbar ist. Wenn man jedoch das TransformGroup Objekt verwenden würde um das ColorCube Objekt zu verändern, wäre es möglich den ColorCube aus der Sicht zu bewegen. Da das Grenzen Objekt zusammen mit dem Verhalten Objekt in der Szene steht, sollte es möglich sein das Objekt weiterhin zu verschieben. Solange das Aktivierungsvolumen einer Sicht und die Scheduling Grenzen für das Verhalten noch schneidet ist das Verhalten weiterhin aktiv. Interaktion mit einem Objekt das außerhalb der Sicht liegt ist sicher nicht schlecht, sofern das gewünscht ist. Das Problem liegt bei der Veränderung der Ansicht. Da sich das Aktivierungsvolumen nicht mehr mit den Scheduling Grenzen schneidet, auch wenn es das visuelle Objekt umfasst, ist das Verhalten nicht mehr aktiv. Dadurch ist das visuelle Objekt mit dem man in der Ansicht interagieren möchte nicht mehr aktiv. Die meisten Benutzer werden dies für ein Problem halten, auch wenn es beabsichtigt ist. Es gibt zwei Lösungen für dieses Problem. Eines ist die Veränderung des Szenengraphen so, dass er die Scheduling Grenzen und das visuellen Objekt beinhaltet. Dies ist leicht zu bewerkstelligen wie in der folgenden Abbildung gezeigt. Die alternative Lösung nutzt ein BoundingLeaf Objekt für die Scheduling Grenzen, auf die ich hier aber nicht näher eingehen werde.



Alternative Szenengraph Abbildung

4.5 Verhaltensklasse API

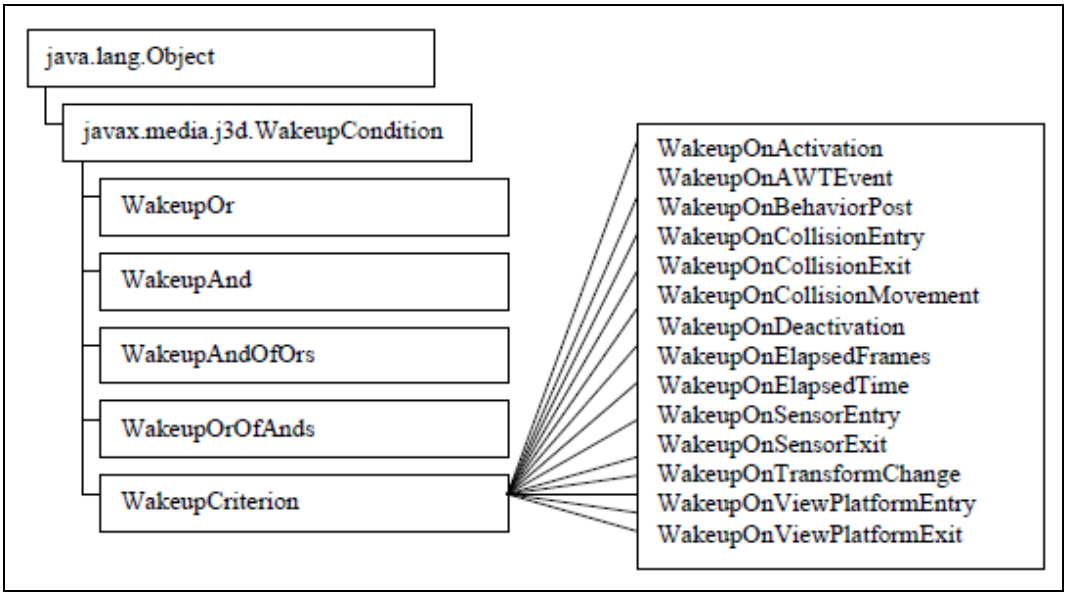
Die folgende Abbildung zeigt die Hierarchie einschließlich der Verhaltensklasse. Es können eigene benutzerspezifische Verhaltensklassen geschrieben werden. Darüber hinaus gibt es viele bestehende Verhaltensweisen in Java 3D API Werkzeugs Paketen. Als Erweiterung eines Blattes können Instanzen einer Klasse, welche Verhalten erweitern, die Kinder einer Gruppe in einem Szenengraphen sein.



Klassenhierarchie eines Verhaltens

4.6 WakeUp Bedingungen

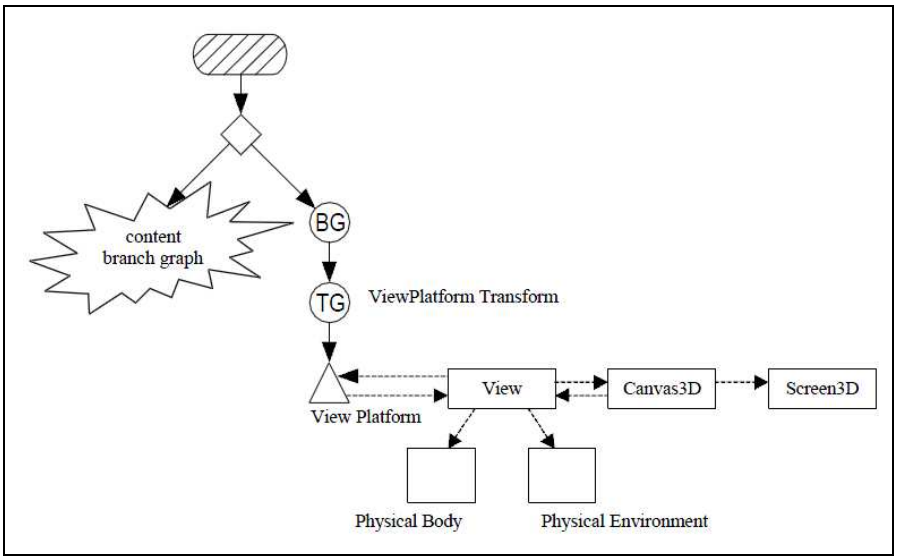
Aktives Verhalten wird durch das Auftreten eines bestimmten Ereignis oder einem WakeUp Ereignis ausgelöst. Der WakeUp Impuls für ein Verhalten nutzen die WakeupCondition Klasse. Die Abstrakte Klasse, WakeupCondition ist die Basis aller WakeUp Klassen in der Java 3D API Hierarchie. Fünf Klassen erweitert die WakeupCondition. Eine ist die Abstrakte Klasse WakeupCriterion. Die andern vier erlauben die Zusammensetzung mehrerer Wakeup Bedingungen in einem einzigen Wakeup Zustand.



Klassenhierarchie WakeupCondition

5. Verhaltensklassen für Tastatur Navigation

Bisher stand der Betrachter in einem festen Standort mit einer festen Orientierung. Die Möglichkeit den Betrachter zu bewegen ist eine wichtige Fähigkeit in vielen 3D-Grafik Anwendungen. Java 3D ist in der Lage den Betrachter zu bewegen. Natürlich gibt es bereits Klassen in Java 3D, die diese Funktionalitäten enthalten und die nur richtig implementiert werden müssen.
Wenn die Transformation geändert wurde, muss man den Betrachter verschieben oder neu orientieren. Es kann auch beides von Nöten sein. Das Standard Design für eine Tastatur Navigation ist recht einfach. Wenn man ein Verhaltensobjekt hat, ändert man einfach die Ansichts-Plattformumwandlung (ViewPlatformTransform) so, dass diese auf eine Tastatureingabe reagiert. Selbstverständlich kann man seine eigene Verhaltensklasse für Tastatur Navigation erstellen.



Standard Ansichts-Teilbaumgraph eines virtuelles Universum in Java 3D

5.1 Navigation in einem Simplen Universum

Ein einfaches Universum und die damit vorhandenen Klassen unterstützen eine Kombination aus Methoden um das Ansichts-Plattformumwandlung Objekt aufzurufen. Dadurch hat man sein eigenes einfaches Universum und kann zusätzlich noch darin navigieren.
Die folgende Codezeile ruft die Ansichts-Plattformumwandlung von einem einfachen Universum Objekt „su“ auf:

```
TransformGroup vpt = su.getViewingPlatform().getViewPlatformTransform();
```

5.1.1 Einfaches Programm für ein Tastatureingabeverhalten

Hier zeige ich, anhand von dem Programm “KeyNavigatorApp“, wie man das Tastatureingabeverhalten anwendet. In diesem Programm kann man gut sehen, dass die Schritte bei der Verwendung einer Tastatur Navigation Verhaltensklasse im Wesentlichen identisch mit denen jeder anderen Verhaltensklasse sind.

1. Tastatur Navigation Verhaltensobjekt erzeugen und Umwandlungsgruppe setzen
2. Tastatur Navigation Verhaltensobjekt dem Szenengraphen hinzufügen
3. Gültigkeitsbereich für das Tastatur Navigation Verhaltensobjekt festlegen

Ruft die Ansichts-Plattformumwandlung von einem einfachen Universum Objekt auf und weist es dem Objekt „vpTrans“ zu.

```
vpTrans = su.getViewingPlatform().getViewPlatformTransform();
```

Erzeugt ein Tastatureingabeverhaltensklasse Objekt und setzt die Umwandlungsgruppe.

```
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehaviour(vpTrans);
```

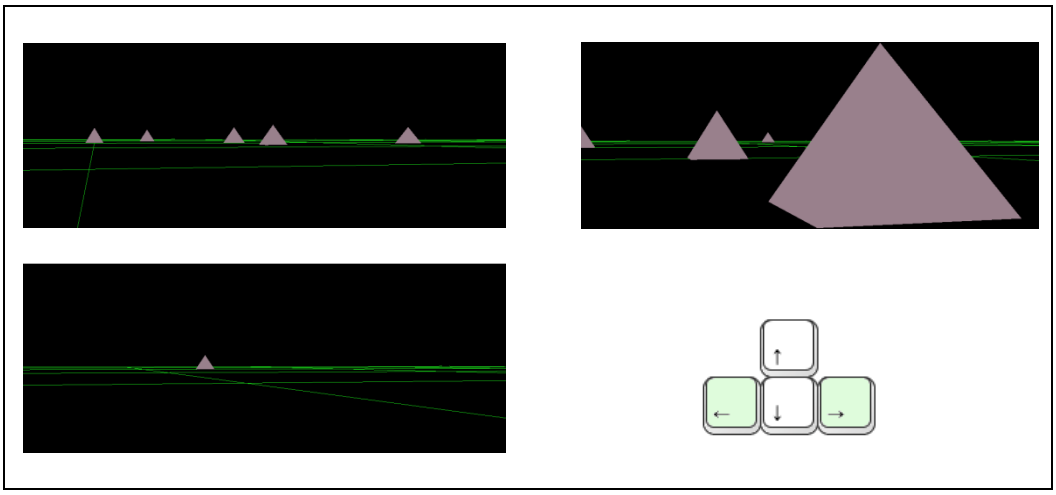
Gültigkeitsbereich für das Tastatureingabeverhaltensklasse Objekt festlegen.

```
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
```

Das Tastatureingabeverhaltensklasse Objekt dem Ansichts-Teilgraph hinzufügen.

```
objRoot.addChild(keyNavBeh);
```

KeyNavigatorApp: *Den Gesamten Source-Code finden sie im Anhang.*



Java 3D Programm mit dem man mit Hilfe der Pfeiltasten navigieren kann.

Mit dem hinzufügen des einfachen Universum Objekt zu der erzeugten Szenengraphen Methode ist es möglich Zugang zu anderen Features des Ansichts-Teilbaumgraphen von dem einfachen Universum zu erhalten. Dies kann z.B. eine Plattform Geometrie, Ansichts-Avatare oder das hinzufügen eines Gültigkeitsbereichs zu dem Ansichts-Teilbaumgraph sein.

5.2 Universelle Applikationen eines Verhaltens

Wie bei jedem Verhaltensobjekt, ist das Tastatur Navigation Verhalten Objekt nur aktiv, wenn die Gültigkeitsgrenzen (scheduling bounds) das Aktivierungsvolumen der Ansichts-Plattform schneidet. Dies kann vor allem das Verhalten der Navigation einschränken. Deshalb sollte das Verhalten immer eingeschaltet sein.

5.2.1 Tastatur Navigation Verhalten und Tastatur Navigation Klassen

Das Tastatur Navigation Werkzeug ist mit Hilfe von zwei Klassen implementiert. Zur Laufzeit gibt es zwei Objekte. Das erste Objekt ist das Tastatur Navigation Verhaltens Objekt und das zweite ist das Tastatur Navigation Objekt. Die zweite Klasse ist hier nicht dokumentiert, da der Programmierer oder der Anwender nicht wissen müssen, dass eine zweite Klasse oder ein zweites Objekt existiert. Die Tastatur Navigation Klasse nimmt das AWT Ereignis und

verarbeitet es bis zur einzelnen Tastendruck Ebene. Die folgende Tabelle zeigt die Wirkung der einzelnen Tastendrücke. Tastatur Navigation implementiert Bewegung mit Beschleunigung.

Taste	Wirkung	Alt-Taste Auswirkung
←	Rotation Links	Seitliche Verschiebung Links
→	Rotation Rechts	Seitliche Verschiebung Rechts
↑	Bewegung vorwärts	
↓	Bewegung rückwärts	
Bild↑	Rotation hoch	Verschiebung hoch
Bild↓	Rotation runter	Verschiebung runter
+	Wiederherstellen des „back clip“ Abstands	
-	Zurücksetzen des „back clip“ Anstands	
=	Zurück zum Mittelpunkt des Universum	

Die folgende Referenz Blöcke zeigen die API für die Tastatur Navigation Verhaltens Werkzeug Klasse.

5.2.2 Tastatur Navigation Verhalten Zusammenfassung

Tastatur Navigation Verhalten Konstruktor Zusammenfassung

Paket: *com.sun.j3d.utils.behaviors.keyboard*
Erweitert: *Behavior*

Diese Klasse ruft die Tastatur Navigation Verhaltensklasse auf, um die Ansichts-Plattformumwandlung zu verändern.

KeyNavigatorBehavior(TransformGroup targetTG)
Erstellt einen neuen Tastatur Navigation Verhaltensknoten der auf der spezifischen Umwandlungsgruppe arbeitet.

Tastatur Navigation Verhalten Methode Zusammenfassung

void initialize()
Überschreibt die Verhaltens Initialisierungsmethode um die aufwach Kriterien zu setzen

void processStimulus(java.util.Enumeration criteria)
Überschreibt die Verhaltens Stimulus-Methode um das Event zu verarbeiten

6. Verhaltensklassen für Maus Interaktion

Das Maus Werkzeug Verhaltens Paket (com.sun.j3d.utils.behaviours.mouse) enthält Verhaltens Klassen in welchen die Maus als Input für Interaktionen mit visuellen Objekten benutzt wird. Inbegriffen sind Klassen für Umwandlung, Zoomen und das rotieren von visuellen Objekten durch die Reaktion der Mausbewegungen.

Die nachfolgende Tabelle zeigt die 3 spezifischen Maus Verhaltensklassen die in dem Paket enthalten sind. Zusätzlich zu diesen 3 Klassen, gibt es eine abstrakte Maus Verhaltensklasse und eine Maus Rückruf Schnittstelle in dem Maus Verhaltens Paket. Diese abstrakte Klasse und die Schnittstelle werden dazu benutzt um eine spezifische Maus Verhaltensklasse zu erstellen und sind nützlich für die Erstellung eines eigenen Mausverhaltens.

Klasse	Wirkung	Mausaktion
MouseRotate	rotiert ein visuelles Objekt	Linke Maustaste halten und bewegen
MouseTranslate	verschiebt ein visuelles Objekt	Rechte Maustaste halten und bewegen
MouseZoom	zoomt ein visuelles Objekt	Mittlere Maustaste halten und bewegen

6.1 Maus Verhaltensklassen benutzen

Die spezifischen Maus Verhaltensklassen sind sehr einfach zu benutzen. Auch hier ist die Nutzung der Klasse im Wesentlichen das gleiche wie bei jeder anderen Verhaltens Klasse.

1. Bereitstellen des Lese- und Schreibzugriffes für die Ziel-Umwandlungsgruppe
2. Mausverhalten Objekt erzeugen
3. Setzen der Ziel-Umwandlungsgruppe
4. Gültigkeitsbereich für das Mausverhalten Objekt festlegen
5. Dem Szenengraphen das Mausverhalten Objekt hinzufügen

Wie bei den meisten Rezepten, müssen die Schritte nicht streng in der vorgegebenen Reihenfolge durchgeführt werden. Der zweite Schritt muss vor dem dritten Schritt, und der vierte Schritt vor dem fünften Schritt ausgeführt werden. Die anderen Schritte können nach Lust und Laune ausgeführt werden. Außerdem kann der zweite und dritte Schritt kombiniert in einem anderen Konstruktor aufgerufen werden.

Setzt die Lese- und Schreibrechte für das Transformationsgruppen Objekt

```
objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
```

Erzeugt ein Mausverhaltensklasse Objekt.

```
MouseRotate myMouseRotate = new MouseRotate();
```

Setzt das Transformgruppen Objekt auf das Mausverhaltensklasse Objekt

```
myMouseRotate.setTransformGroup(objRotate);
```

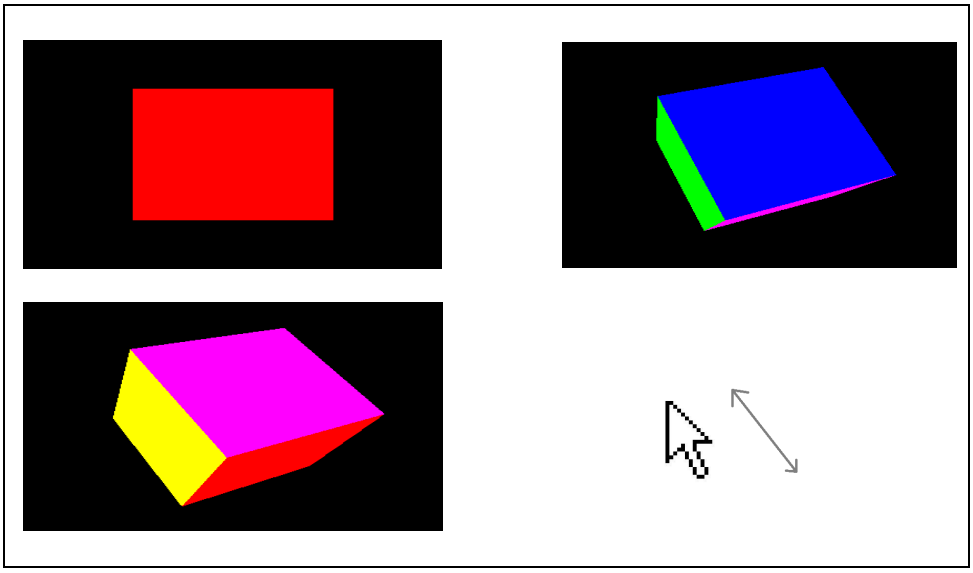
Gültigkeitsbereich für das Mausverhaltensklasse Objekt festlegen.

```
myMouseRotate.setSchedulingBounds(new BoundingSphere());
```

Das Mausverhaltensklasse Objekt der Transformationsgruppe hinzufügen.

```
objRotate.addChild(myMouseRotate);
```

MouseRotateApp: *Den Gesamten Source-Code finden sie im Anhang.*



Java 3D Programm mit dem man mit Hilfe der Maus das Objekt rotieren kann.

Das gleiche Beispiel funktioniert auch mit den andern zwei Mausverhaltensklassen. Es ist auch möglich alle drei Verhaltensweisen in der gleiche Anwendung und dem gleichen visuellen Objekt zu verwenden. Wenn das Mausverhalten zuerst gelesen und in das Ziel geschrieben wird, wird nur ein Transformgruppenobjekt für alle 3 Verhaltensklassenobjekte benötigt. Jedes Objekt das sie nun der Transformgruppe hinzufügen reagiert nun auf das Verhalten ihrer Mausverhaltensklasse.
Wenn es nicht gewünscht ist, dass mehrere Objekte das gleiche Mausverhalten aufweisen, gibt es zwei Möglichkeiten um dies zu ändern.

1. Die Position des Betrachters ändern oder den Gültigkeitsbereich des Verhaltens (scheduling bounds) so ändern, dass nur ein Verhalten aktiv ist.
2. Verwendung eines Picking-Mechanismus um das Verhalten zu isolieren. Darauf werde ich später noch genauer darauf eingehen.

6.1.1 Maus Verhalten Zusammenfassung

Maus Rotation Konstruktor Zusammenfassung

Paket: *com.sun.j3d.utils.behaviors.mouse*
Erweitert: Mouse*Behavior*

Diese Klasse ermöglicht das Verhalten durch Rotation eines Objektes durch Mausdruck und -bewegung zu beeinflussen.

MouseRotate()
Erstellt ein Standard Maus Rotationsverhalten.

MouseRotate(TransformGroup transformGroup)
Erstellt ein Maus Rotationsverhalten auf die übergebene Transformationsgruppe.

MouseRotate(int flags)
Erstellt ein Maus Rotationsverhalten mit den gesetzten Flags

Maus Translate Konstruktor Zusammenfassung

Paket: *com.sun.j3d.utils.behaviors.mouse*
Erweitert: Mouse*Behavior*

Diese Klasse ermöglicht das Verhalten durch Verschiebung auf der X- und Y-Achse eines Objektes durch Mausdruck und -bewegung zu beeinflussen.

MouseTranslate()
Erstellt ein Standard Maus Verschiebung verhalten.

MouseTranslate(TransformGroup transformGroup)
Erstellt ein Maus Verschiebung Verhalten auf die übergebene Transformationsgruppe.

MouseTranslate(int flags)
Erstellt ein Maus Verschiebung Verhalten mit den gesetzten Flags

Maus Zoom Konstruktor Zusammenfassung

Paket: *com.sun.j3d.utils.behaviors.mouse*
Erweitert: Mouse*Behavior*

Diese Klasse ermöglicht das Verhalten durch Verschiebung auf der Z-Achse (Zoom) eines Objektes durch Mausdruck und -bewegung zu beeinflussen.

MouseTranslate()
Erstellt ein Standard Maus Zoomverhalten.

MouseTranslate(TransformGroup transformGroup)
Erstellt ein Maus Zoomverhalten auf die übergebene Transformationsgruppe.

MouseTranslate(int flags)
Erstellt ein Maus Zoomverhalten mit den gesetzten Flags

Maus Rotate/Translate/Zoom Methode Zusammenfassung

void setFactor(double factor)
Setzt die X- und Y-Achsenbewegungsmultiplikator mit einem Faktor.

void setFactor(double xFactor, double yFactor) *(Nicht bei MouseZoom)*
Setzt die X- und Y-Achsenbewegungsmultiplikator mit xFaktor und yFaktor.

void setCallback(MouseBehaviorCallback callback)
Die Transformveränderungsmethode in der Callback Klasse wird jedes Mal aufgerufen, wenn eine Veränderung aktualisiert wird.

void transformChanged(Transform3D transform)
Der Benutzer kann diese Methode, die jedes Mal aufgerufen wird, wenn eine Veränderung aktualisiert wird, überladen. Die Default-Implementierung hat keine Wirkung.

6.1.2 Eigene Maus Verhaltensklassen

Das spezifische Verhalten der Maus Klassen (MouseRotate, MouseTranslate und MouseZoom) sind Erweiterungen der abstrakten Maus Verhaltensklasse und implementieren die Maus Callback Schnittstelle.
Wenn man seine eigene Maus-Verhaltensklasse schreiben möchte, muss diese abstrakte Klasse erweitert werden. Die setTransformGroup() Methode ist vermutlich die einzige Methode, einer Instanz des Mausverhaltens, die vom Benutzer benutzt wird. Die anderen Methoden sind für die Programmierer von benutzerspezifischen Maus-Verhaltensklassen gedacht.

Maus Verhalten Methode Zusammenfassung

void initialize()
Initialisiert das Verhalten.

void processMouseEvent(java.awt.event.MouseEvent evt)
Verarbeitet das Mausgeschehen.

void processStimulus (java.util.Enumeration criteria)
Alle Mausmanipulationen müssen diese Methode des Verhaltens implementieren.

void setTransformGroup(TransformGroup transformGroup)
Erstellt ein Verhalten auf die übergebene Transformationsgruppe.

void wakeup ()
Manuelles aufwecken des Verhaltens.

6.1.3 Maus Callback Schnittstelle

Eine Klasse implementiert diese Schnittstelle welche die die transformChanged-Methode liefert, die aufgerufen wird, wenn das Zielobjekt in irgendwelcher Hinsicht geändert wurde. Jede dieser 3 spezifischen Maus-Verhaltens Klassen implementiert diese Klasse. Ein Programmierer kann einfach die transformChanged Methode überschreiben, um eine Methode zu spezifizieren, nachdem ein Transformation Objekt geändert wurde.

Schnittstelle Maus Verhalten Callback Methode Zusammenfassung

Paket: *com.sun.j3d.utils.behaviors.mouse*

void transformChanged(int type, Transform3D transform)
Klassen implementieren diese Schnittstelle, welche mit einer der Maus Verhaltensklassen registriert sind, die jedes Mal aufgerufen werden, wenn das Verhalten geändert wurde.

7. Picking

Picking gibt dem Benutzer die Möglichkeit, mit einzelnen visuellen Objekten in der Szene zu interagieren. Picking ist als ein typisches Verhalten eines Maus-Events implementiert. Beim Picking eines visuellen Objektes, fährt der Benutzer den Mauszeiger über das visuelle Objekt

seiner Wahl und drückt die Maustaste. Sobald die Taste gedrückt wird, beginnt die Picking-Operation. Ein Strahl wird in die virtuelle Welt von der Position des Mauszeigers parallel mit der Projektion projiziert. Anschließen wird der Schnittpunkt zwischen diesem Strahl und dem Objekt in der virtuellen Welt berechnet. Das visuelle Objekt das sich am nächsten zur Bild Platte schneidet ist für die Interaktion ausgewählt.

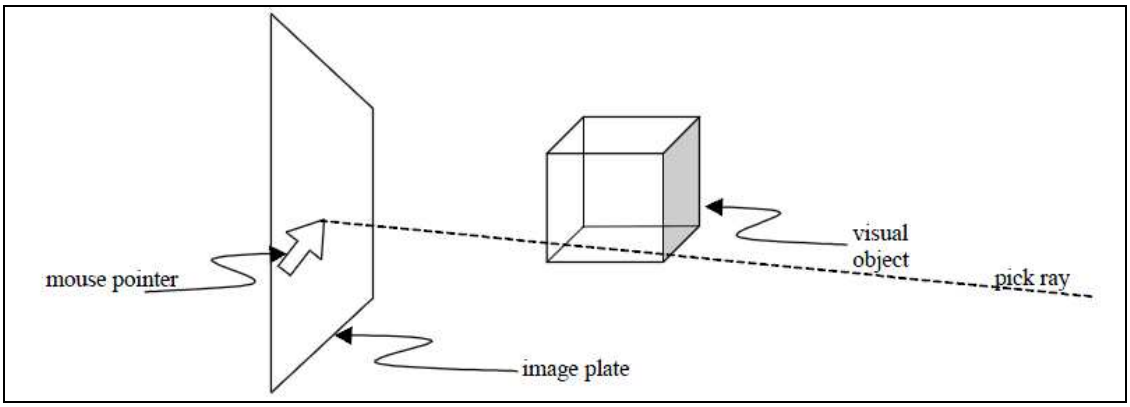
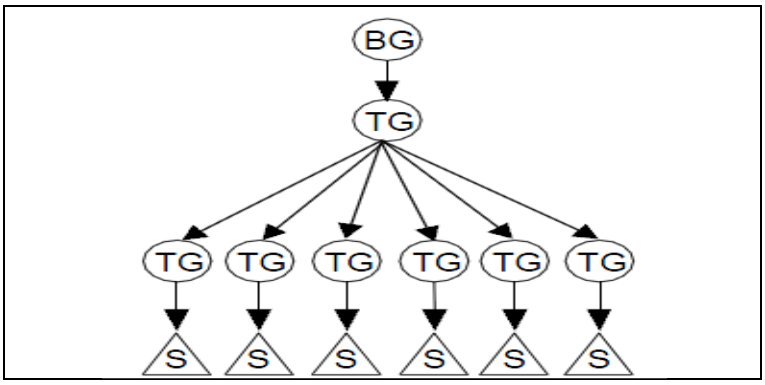


Abbildung wie ein Picking-Strahl in einer virtuellen Welt projiziert wird.

In manchen Fällen findet die Interaktion nicht direkt mit dem ausgewählten Objekt statt, sondern mit einem Objekt entlang des Szenengraphen-Pfad. Zum Beispiel, wenn ein Würfel-Objekt für Rotation ausgewählt wird, wird das Würfel-Objekt nicht manipuliert, da das Umwandlungsgruppen Objekt oberhalb des Würfel-Objekts im Szenengraph-Pfad ist. Allerdings, wenn ein visuelles Objekt per Picking ausgewählt wird und die Farbe verändern soll, dann wird dafür das visuelle Objekt benötigt.
Die Bestimmung des Gegenstandes für die weitere Bearbeitung ist nicht immer einfach.
Wenn ein visuelles Objekt (z.B. Würfel) gedreht werden soll, das aus sechs einzelnen Shape-Objekten und sechs Umwandlungsgruppen-Objekten zusammensetzt ist, wie in der unteren Abbildung zu sehen ist, ist es nicht das Umwandlungsgruppen-Objekt über den einzelnen Shapes das geändert werden muss. Dieser Würfel muss unterhalb der Umwandlungsgruppe des Teilbaumgruppen-Objekt im Szenengraph manipuliert werden. Aus diesem Grund sollte das Ergebnis aus mehreren Picking-Operationen den Szenengraphen-Pfad für die weitere Verarbeitung zurückliefern.



Szenengraph Diagramm für einen Würfel bestehen aus 6 Shapes

Die Schnittpunkt-Prüfung ist sehr rechenintensiv. Deshalb ist Picking rechenintensiv und umso mehr die Komplexität der Szene zunimmt, umso aufwendiger wird die Berechnung. Die Java 3-D API bietet eine Reihe von Möglichkeiten an, z.B. kann der Programmierer die

Menge der Berechnungen in Picking festlegen. Ein wichtiger Punkt sind die Fähigkeiten und Attribute von Szenengraph-Knoten. Ein Szenengraph-Knoten muss aufsammelbar (pickable) sein und somit muss auch die setPickable() Methode von der Klasse gesetzt sein. Wenn ein Knoten mit setPickable() auf false gesetzt ist, ist dieser Knoten und seine Kinder nicht aufsammelbar. Folglich werden diese Knoten bei der Berechnung nicht berücksichtigt.
Eine weitere Picking unterstützte Erweiterung von der Knotenklasse ist die ENABLE_PICK_REPORTING Fähigkeit. Diese Fähigkeit ist nur bei Gruppenknoten verfügbar. Wenn diese für eine Gruppe gesetzt wird, ist das Gruppenobjekt immer im Szenengraph-Pfad inbegriffen und wird bei einem Pick-Vorgang zurückgeben. Gruppenknoten werden in einem Szenengraphen-Pfad ausgeschlossen, wenn die Fähigkeit nicht eingestellt ist. Wenn nicht die richtigen Einstellungen für den Szenengraph-Knoten eingestellt sind, kann das sehr frustrierend für den Entwickler von Applikation mit Picking-Operation sein. Die folgenden 2 Referenzblöcke zeigen jeweils die Knoten-Methoden und Fähigkeiten.

Knoten Methode (Auszug)

Paket: *SceneGraphObject*
Erweitert: Group, Leaf

Die Knoten-Klasse stellt eine abstrakte Klasse für alle Gruppen- und Blattknoten. Sie bietet einen gemeinsamen Rahmen für den Aufbau eines 3D Szenengraphen, speziell Hüllvolumen, Picking und Kollisions Fähigkeiten.

void setBounds(Bounds bounds)
Setzt die geometrischen Grenzen eines Knotens

void setBoundsAutoCompute(boolean autoCompute)
Automatische Kalkulation geometrischer Grenzen eines Knoten an- / ausschalten.

setPickable(boolean pickable)
True ist der Knoten aufsammelbar, false ist er + seine Kinder nicht aufsammelbar.

Knoten Fähigkeiten (Auszug)

ENABLE PICK REPORTING
Dieser Knoten wird zurück gegeben, wenn im Szenengraph-Pfad Picking auftritt. Diese Fähigkeit ist nur für Gruppen verfügbar, Blattknoten werden ignoriert. Die Standardeinstellung für den Gruppenknoten ist auf false gesetzt.

ALLOW_BOUNDS_READ | WRITE
Gibt an, dass es bei diesem Knoten erlaubt ist die Grenzinformationen zu lesen (schreiben).

ALLOW_PICKABLE_READ | WRITE
Gibt an, dass es bei diesem Knoten erlaubt den Pick-Zustand zu lesen (schreiben).

Ein anderer Weg den ein Programmierer machen kann um die Berechnungen für das Picking zu verkleinern, ist das prüfen von kreuzenden Grenzen anstelle von geometrischen Schnittpunkten. Mehrere der Picking verwandten Klassen haben Konstruktoren und/oder Methoden mit Paramatern mit der sie festlegen können, was benutzt werden soll: USE_BOUNDS oder USE_GEOMETRY. Wenn die USE_BOUNDS Option ausgewählt ist, wird

das Picking anhand der Grenzen des visuellen Objektes bestimmt, nicht die tatsächliche Geometrie. Die Bestimmung des Pickings über die Grenzen ist wesentlich einfacher, und somit ergibt sich eine bessere Leistung für die einfachsten geometrischen Formen. Natürlich ist der Nachteil, dass die Bestimmung der Grenzen nicht so präzise ist.
Eine dritte Programmiertechnik zur Reduzierung des Rechenaufwandes ist die Tragweite der Picking Prüfung auf den maßgeblichen Teil des Szenengraphen zu begrenzen. In jeder Picking Werkzeugs-Klasse wird der Knoten des Szenengrapen als die Wurzel des Graphen für die Picking-Prüfung gesetzt. Dieser Knoten ist nicht unbedingt die Wurzel des Inhalts-Teilbaumgraphen. Im Gegenteil, der Knoten sollte die Wurzel des Inhalts-Untergraphen sein, welcher nur die aufsammelbaren Objekte enthält, sofern dies möglich ist. Diese Überlegung kann ein wichtiger Faktor bei der Bestimmung des Aufbaus des Szenengraphen für einige Anwendungen sein.

7.1 Picking Werkzeuge benutzen

Es gibt zwei grundlegende Ansätze zur Verwendung von Picking Funktionen in Java 3D:
Verwendung der Objekte der Picking-Klassen, oder das erstellen einer eigenen Picking-Klasse und das verwenden von Instanzen dieser Klasse. Die Picking Pakete enthalten Klassen für pick/Rotation, pick/Verschiebung und Pick/Zoom. Das heißt, dass ein Benutzer sein Objekt drehen kann, indem er den Mauszeige über das gewünschte Objekt platziert, den Mauszeiger drückt und die Maus nach unten zieht. Jeder der Picking-Klassen verwendet eine andere Maustaste. Es ist aber auch möglich alle 3 Picking-Klassen für ein Objekt zu benutzen.
Da ein Picking Verhalten auf jedem Szenengraphen Objekt funktioniert, muss nur ein Picking Verhaltensobjekt bereitgestellt werden. Nur die folgenden 2 Zeilen Code werden benötigt um Picking in Java 3D Programmen einzubinden:

```
PickRotateBehavior behavior = new PickRotateBehavior(root, canvas, bounds);
root.addChild(behavior);
```

Das obige Verhaltensobjekt wird für jedes Picking Event auf dem Szenengraph gesetzt und behandelt die Mausdrücke. Die Wurzel des Szenengraphen stellt den Teil für die Prüfung des Picking, die Leinwand ist der Ort wo die Maus sich befindet und die Grenzen sind durch die Scheduling Grenzen von dem Picking Verhaltensobjekt festgelegt.

1. Erstelle den Szenengraph
2. Erzeuge ein Picking Verhaltensobjekt mit der Wurzel, Leinwand und Grenzspezifikationen
3. Füge das Verhaltensobjekt dem Szenengraph hinzu
4. Aktiviere die entsprechenden Fähigkeiten für das Szenengraph Objekt

Setze die entsprechenden Fähigkeit für das Szenengraphen Objekt.

```
objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
objRotate.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
```

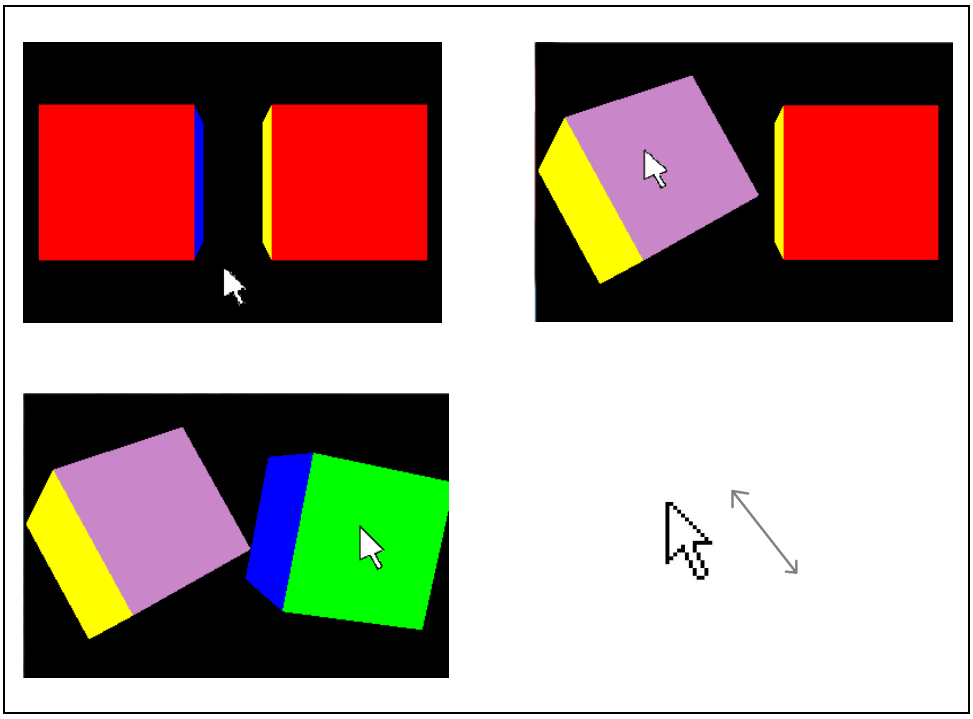
Erzeuge das Picking Verhaltensobjekt mit dem Wurzelknoten der Leinwand und den Grenzspezifikationen.

```
pickRotate = new PickRotateBehavior(objRoot, canvas, behaveBounds);
```

Picking Verhaltensobjekt dem Szenengraph hinzufügen.

```
objRoot.addChild(pickRotate);
```

MousePickApp: *Den Gesamten Source-Code finden sie im Anhang.*



Java 3D Programm mit dem man mit Hilfe der Maus bestimmte Objekt rotieren kann.

7.2 Java 3D API Haupt Picking Klassen

Es gibt drei Ebenen von Picking Klassen welche in Java 3D zur Verfügung stehen. Der Java 3D API Kern stellt die niedrigste Level-Funktionalität. Die Picking Werkzeug Pakete enthalten die allgemeinen Verhaltensklassen, geeignet zur Anpassung. Die Picking Werkzeug Klasse bietet auch spezielle Klassen, die das Picking Verhalten direkt in Java 3D Programmen verwenden.
Die allgemeine Picking Werkzeug Klasse kombiniert grundlegende Picking Operationen in Verhaltensobjekten. Die spezifische Picking Werkzeug Klassen nutzen die allgemeinen Klassen um spezifisches Picking Verhalten zu implementieren.

7.2.1 PickShape Klassen

Diese abstrackte Klasse stellt weder Konstruktoren noch Methoden zur Verfügung. Sie bietet Abstraktion für vier Unterklassen:

- **PickBounds**
  Objekte von der Klasse PickBounds stellen einen Rahmen für Pick Tests.
- **PickRay**
  Objekte von der Klasse PickRay repräsentieren einen Strahl (ein Punkt und eine Richtung) für Picking.
- **PickSegment**
  Objekte von der Klasse PickSegment repräsentieren eine Strecke (definiert durch zwei Punkte) für Picking.

- **PickPoint**
  Objekte von der Klasse PickPoint repräsentieren einen Punkt für Picking.

Spezifische Picking Verhaltens Klassen

- **PickRotateBehavior**
  Die PickRotateBehavior erlaubt dem Benutzer das visuelle Objekt interaktiv auszuwählen und zu drehen.
- **PickTranslateBehavior**
  Die PickTranslateBehavior erlaubt dem Benutzer das visuelle Objekt interaktiv auszuwählen und zu verschieben.
- **PickZoomBehavior**
  Die PickZoomBehavior erlaubt dem Benutzer das visuelle Objekt interaktiv auszuwählen und zu zoomen.

8. Fazit

Ich hoffe dass ich mit diesem Java 3D Tutorial jedem einen Einblick in die Interaktion mit Java 3D geben und zeigen konnte. Durch die einfache Installation, die gute Java 3D Dokumentation und Tutorials aus dem Internet ist es für Java-Entwickler sehr einfach Java 3D einzubinden und damit zu arbeiten. Bei der Installation ist darauf zu achten, dass man die richtige Bit-Version benutzt, d.h. für ein 32-Bit System auch die 32-Bit Java 3D Version benutzt, sonst kann dies zu Komplikationen führen. Bei der Programmierung selbst traten keine nennenswerten Schwierigkeiten auf. Gerne hätte ich noch ein eigenes Beispiel programmiert, das alle Interaktionen vereint, jedoch haben leider Zeit und Umfang der Ausarbeitung dies verhindert.

9. Anhang

9.1 Softwareressourcen

Die Programmierung und die Ausarbeitung erfolgte an einem normal Arbeitsplatz-Rechner mit installierter Java 3D Entwicklungsumgebung.

Software

- Eclipse mit Java 3D Plug-in - Java 3D Entwicklung
- Microsoft Word - Tutorial
- Open Office - Präsentation

9.2 Quellen

Dieses Tutorial baut auf das Englische Tutorial von Java 3D - Chapter 4 auf. Der Großteil der hier verwendeten Bilder wurde aus diesem Tutorial entnommen.

Quellen

- java.net
  https://j3d-core.dev.java.net/
- Java 3D Tutorial - Chapter 4
  http://java.sun.com/developer/onlineTraining/java3d/j3d_tutorial_ch4.pdf
- Java 3D Dokumentation
  http://download.java.net/media/java3d/javadoc/1.5.2/index.html
- Source Code
  http://java.sun.com/developer/onlineTraining/java3d/

9.3 Source-Code

SimpleBehaviorApp

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import java.awt.event.*;
import java.util.Enumeration;

public class SimpleBehaviorApp extends Applet {
    public class SimpleBehavior extends Behavior{
        private TransformGroup targetTG;
        private Transform3D rotation = new Transform3D();
        private double angle = 0.0;

        SimpleBehavior(TransformGroup targetTG){
            this.targetTG = targetTG;
        }

        public void initialize(){
            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        }

        public void processStimulus(Enumeration criteria){
            angle += 0.1;
            rotation.rotY(angle);
```

```
            targetTG.setTransform(rotation);
            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        }
    }

    public BranchGroup createSceneGraph() {
       BranchGroup objRoot = new BranchGroup();

        TransformGroup objRotate = new TransformGroup();
        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

       objRoot.addChild(objRotate);
       objRotate.addChild(new ColorCube(0.4));

        SimpleBehavior myRotationBehavior = new SimpleBehavior(objRotate);
        myRotationBehavior.setSchedulingBounds(new BoundingSphere());
        objRoot.addChild(myRotationBehavior);

        objRoot.compile();

      return objRoot;
    }

    public SimpleBehaviorApp() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config =
           SimpleUniverse.getPreferredConfiguration();

        Canvas3D canvas3D = new Canvas3D(config);
        add("Center", canvas3D);

        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewingPlatform().setNominalViewingTransform();
        simpleU.addBranchGraph(scene);
    }

    public static void main(String[] args) {
        new MainFrame(new SimpleBehaviorApp(), 256, 256);
    }
}
```

KeyNavigatorApp

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class KeyNavigatorApp extends Applet {
    Shape3D createPyramid(){
        IndexedTriangleArray pyGeom =
                new IndexedTriangleArray(5,  GeometryArray.COORDINATES
                                           | GeometryArray.COLOR_3
                                          , 12);

        pyGeom.setCoordinate(0, new Point3f(  0.0f, 0.7f,  0.0f ));
        pyGeom.setCoordinate(1, new Point3f( -0.4f, 0.0f, -0.4f ));
        pyGeom.setCoordinate(2, new Point3f( -0.4f, 0.0f,  0.4f ));
        pyGeom.setCoordinate(3, new Point3f(  0.4f, 0.0f,  0.4f ));
        pyGeom.setCoordinate(4, new Point3f(  0.4f, 0.0f, -0.4f ));

        pyGeom.setCoordinateIndex( 0, 0);
        pyGeom.setCoordinateIndex( 1, 1);
```

```
        pyGeom.setCoordinateIndex( 2, 2);
        pyGeom.setCoordinateIndex( 3, 0);
        pyGeom.setCoordinateIndex( 4, 2);
        pyGeom.setCoordinateIndex( 5, 3);
        pyGeom.setCoordinateIndex( 6, 0);
        pyGeom.setCoordinateIndex( 7, 3);
        pyGeom.setCoordinateIndex( 8, 4);
        pyGeom.setCoordinateIndex( 9, 0);
        pyGeom.setCoordinateIndex(10, 4);
        pyGeom.setCoordinateIndex(11, 1);

        Color3f c = new Color3f(0.6f, 0.5f, 0.55f);
        pyGeom.setColor(0, c);
        pyGeom.setColor(1, c);
        pyGeom.setColor(2, c);
        pyGeom.setColor(3, c);
        pyGeom.setColor(4, c);

        Shape3D pyramid = new Shape3D(pyGeom);
        return pyramid;
    }

    Shape3D createLand(){
        LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES
                                             | GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int c = 0; c < 44; c+=4){

            landGeom.setCoordinate( c+0, new Point3f( -50.0f, 0.0f,  l ));
            landGeom.setCoordinate( c+1, new Point3f(  50.0f, 0.0f,  l ));
            landGeom.setCoordinate( c+2, new Point3f(   l   , 0.0f, -50.0f ));
            landGeom.setCoordinate( c+3, new Point3f(   l   , 0.0f,  50.0f ));
            l += 10.0f;
        }

        Color3f c = new Color3f(0.1f, 0.8f, 0.1f);
        for(int i = 0; i < 44; i++) landGeom.setColor( i, c);

        return new Shape3D(landGeom);
    }

    public BranchGroup createSceneGraph(SimpleUniverse su) {
        TransformGroup vpTrans = null;

        BranchGroup objRoot = new BranchGroup();

        Vector3f translate = new Vector3f();
        Transform3D T3D = new Transform3D();
        TransformGroup TG = null;

        objRoot.addChild(createLand());

        SharedGroup share = new SharedGroup();
        share.addChild(createPyramid());

        float[][] position = {{  0.0f, 0.0f,  -3.0f},
                              {  6.0f, 0.0f,   0.0f},
                              {  6.0f, 0.0f,   6.0f},
                              {  3.0f, 0.0f, -10.0f},
                              { 13.0f, 0.0f, -30.0f},
                              {-13.0f, 0.0f,  30.0f},
                              {-13.0f, 0.0f,  23.0f},
                              { 13.0f, 0.0f,   3.0f}};

        for (int i = 0; i < position.length; i++){
                translate.set(position[i]);
                T3D.setTranslation(translate);
                TG = new TransformGroup(T3D);
```

```
                TG.addChild(new Link(share));
                objRoot.addChild(TG);
        }
        vpTrans = su.getViewingPlatform().getViewPlatformTransform();
        translate.set( 0.0f, 0.3f, 0.0f);
        T3D.setTranslation(translate);
        vpTrans.setTransform(T3D);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTrans);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        objRoot.addChild(keyNavBeh);

        objRoot.compile();

        return objRoot;
    }

    public KeyNavigatorApp() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config =
           SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        add("Center", canvas3D);
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        BranchGroup scene = createSceneGraph(simpleU);
        simpleU.addBranchGraph(scene);
    }

    public static void main(String[] args) {
        new MainFrame(new KeyNavigatorApp(), 256, 256);
    }
}
```

MouseRotateApp

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import javax.media.j3d.*;

public class MouseRotateApp extends Applet {
       public BranchGroup createSceneGraph() {
              BranchGroup objRoot = new BranchGroup();

              TransformGroup objRotate = new TransformGroup();
              objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
              objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

              objRoot.addChild(objRotate);
              objRotate.addChild(new ColorCube(0.4));

              MouseRotate myMouseRotate = new MouseRotate();
              myMouseRotate.setTransformGroup(objRotate);
              myMouseRotate.setSchedulingBounds(new BoundingSphere());
              objRoot.addChild(myMouseRotate);

              objRoot.compile();

              return objRoot;
       }

       public MouseRotateApp() {
              setLayout(new BorderLayout());
              GraphicsConfiguration config = SimpleUniverse
                           .getPreferredConfiguration();
```

```
            Canvas3D canvas3D = new Canvas3D(config);
            add("Center", canvas3D);

            BranchGroup scene = createSceneGraph();
            SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
            simpleU.getViewingPlatform().setNominalViewingTransform();
            simpleU.addBranchGraph(scene);
      }

      public static void main(String[] args) {
            new MainFrame(new MouseRotateApp(), 256, 256);
      }
}
```

MousePickApp

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.pickfast.behaviors.*;
import javax.media.j3d.*;
import javax.vecmath.*;

public class MousePickApp extends Applet {
    public BranchGroup createSceneGraph(Canvas3D canvas) {
       BranchGroup objRoot = new BranchGroup();

        TransformGroup objRotate = null;
        PickRotateBehavior pickRotate = null;
        Transform3D transform = new Transform3D();
        BoundingSphere behaveBounds = new BoundingSphere();

        transform.setTranslation(new Vector3f(-0.6f, 0.0f, -0.6f));
        objRotate = new TransformGroup(transform);
        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        objRotate.setCapability(TransformGroup.ENABLE_PICK_REPORTING);

        objRoot.addChild(objRotate);
        objRotate.addChild(new ColorCube(0.4));

        pickRotate = new PickRotateBehavior(objRoot, canvas, behaveBounds);
        objRoot.addChild(pickRotate);

        transform.setTranslation(new Vector3f( 0.6f, 0.0f, -0.6f));
        objRotate = new TransformGroup(transform);
        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        objRotate.setCapability(TransformGroup.ENABLE_PICK_REPORTING);

        objRoot.addChild(objRotate);
        objRotate.addChild(new ColorCube(0.4));

        objRoot.compile();

       return objRoot;
    }

    public MousePickApp() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config =
           SimpleUniverse.getPreferredConfiguration();

        Canvas3D canvas3D = new Canvas3D(config);
```

```
        add("Center", canvas3D);

        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

        BranchGroup scene = createSceneGraph(canvas3D);

        simpleU.getViewingPlatform().setNominalViewingTransform();

        simpleU.addBranchGraph(scene);
    }

    public static void main(String[] args) {
        new MainFrame(new MousePickApp(), 256, 256);
    }
}
```