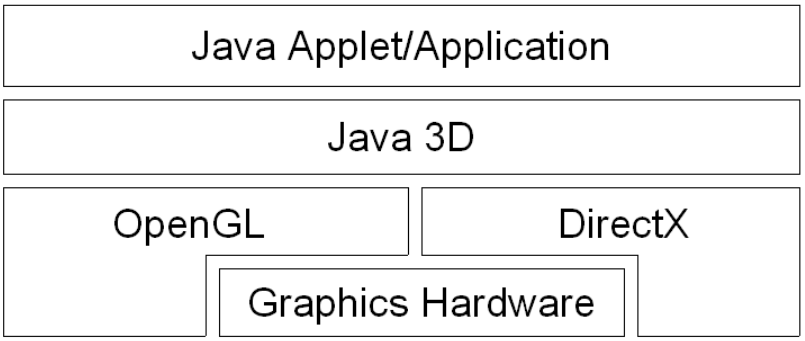
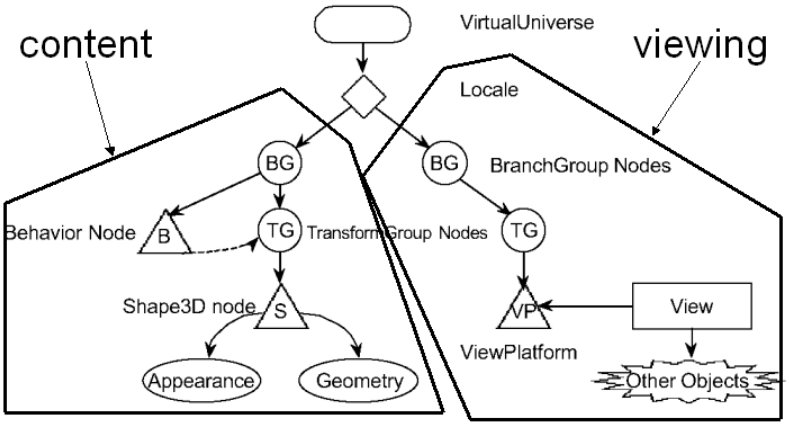
Java 3d

Java3D ist eine Bibliothek von Java Klassen zur Darstellung dreidimensionaler Graphik innerhalb von Java Applikationen und Applets. Die Struktur der darzustellenden Szene wird durch das Aufbauen eines entsprechenden Szenegraphen definiert. Im Szenegraphen wird der logische Aufbau der darzustellenden Objekte auf eine gleichartig aufgebaute, baumähnliche Struktur, die im wesentlichen aus Definitionen von Transformationen und Geometriedaten besteht, abgebildet. Die strukturierte Sicht der Szene erlaubt eine wesentlich komfortablere Handhabung der Objekte (im Vergleich zu OpenGL Renderlists z.B).



Für die Darstellung der Szene setzt Java3D auf ein lokal vorhandenes Rendering-API (OpenGL, unter Windows auch DirectX) auf. Von diesen Schnittstellen verwendete 3D-Beschleunigungshardware wird dadurch auch für das Rendern von Java3D Szenen genutzt. Durch den Zugriff auf lokale, plattformabhängige Renderingschnittstellen, muß für jede Plattform, auf der Java3D benutzt werden soll, eine eigens angepaßte Version installiert werden. Die Schnittstelle zur Application/Applet, die Java3D verwendet bleibt jedoch immer gleich – einmal kompiliert, läuft ein Programm auf verschiedenen Plattformen – vorausgesetzt Java3D ist lokal installiert.
Für die Darstellung wird der Szenegraph ausgewertet, und wenn zulässig intern in eine auf Renderingperformance optimierte Darstellung konvertiert („kompiliert“). Diese Optimierung kann z.B: die Erzeugung von Renderlists für OpenGL und Konvertierung der Datenrepräsentation in eine für das Rendering-API günstige Form sein. Bei Änderungen an der Szene (Geometrie, Transformationen oder Struktur des Graphen) erfolgt eine erneute Auswertung.
Java3D nimmt dem Benutzer einige lästige Aufgaben und Optimierungen ab, die bei direkter Benutzung von z. B. OpenGL durch das Anwendungsprogramm implementiert werden müßten: Das Sortieren von transparenten Polygonen um eine korrekte Darstellung zu ermöglichen, oder räumlich beschränkte Effekte entsprechend ihrem Wirkungsradius aktivieren (Nebel, Lichtquellen, Sound).



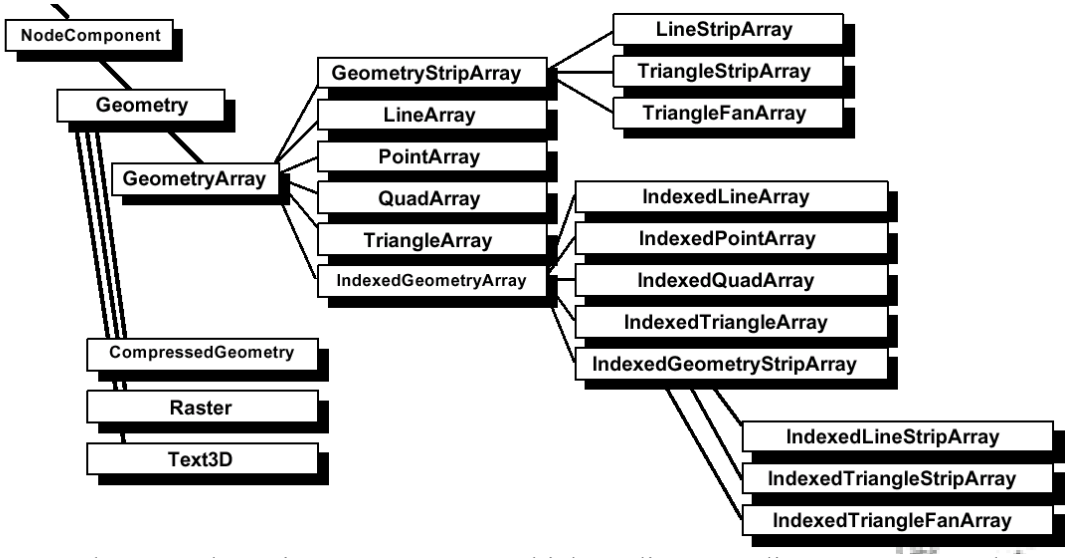
Der Szenegraph

Der Szenegraph von Java3D zerfällt in zwei Teile: den „viewing"-Zweig, der alle für die Definition der Darstellung und Projektion erforderlichen Parameter enthält und den „content"-Zweig, der die darzustellende Szene definiert (Geometrie, Transformationen, …).
Knotenobjekte des Graphen befinden sich immer in einem von drei Zuständen: Nach der Erzeugung, vor dem Einfügen in den Graphen ist ein Knoten „detached" – alle seine Attribute können ausgelesen und verändert werden. Wird der Knoten in den Szenegraphen eingefügt, so ist er „live" – nur Operationen die vorher mittels `setCapability(…)` freigegeben wurden, können durchgeführt werden. Wird der Teilgraph, zu dem ein Knoten gehört mittels `compile()` optimiert, ist der Knoten im zustand „compiled". Die im Knoten enthaltene Information wurde für das Rendering optimiert - Änderungsmöglichkeiten sind dadurch stark eingeschränkt, Geometrieinformation kann z.B. nicht mehr verändert werden.

Im Folgenden eine kurze Beschreibung der wichtigsten Knoten des Szenegraphen:

- *VirtualUniverse* Wurzelknoten eines Szenegraphen.
- *Locale* Enthält Ursprung des darunterhängenden Teiles des Szenegraphen mit hoher Genauigkeit (768 bit). Mehrere Locales werden benötigt um Geometrie mit ausreichender Genauigkeit über mehrere Größenordnungen hinweg zu definieren (z.B: Flug Stadt->Haus->Tür->Tisch->Chip->Schaltkreis). Die meisten Anwendungen kommen jedoch mit nur einem Locale Objekt aus.
- *Group* Vorfahre aller Zwischenknoten im Graphen.
- *TransformGroup(TG)* Enthält eine Transformation, die auf alle seine Nachfolger angewendet wird.
- *BranchGroup(BG)* Gruppiert Nachfolger. BranchGroups können als einziger Knotentyp wieder aus dem Graphen entfernt werden (damit wird der unter der BranchGroup hängende Teilgraph ebenfalls entfernt).
- *Shape3D(S)* Blattknoten, definiert ein Geometrieobjekt in der Szene – Bündelt Geometriedaten, Informationen zum Renderingmodus,

Materialeigenschaften, Textur. Diese Daten werden in eigenen, mit dem Shape3D assoziierten Objekten gespeichert: `Appearance` und `Geometry`. Während die Appearance Klasse Materialeigenschaften und Renderingattribute definiert (Farbe, Reflektionskoeffizienten, Backface Culling, wireframe, …), enthalten von Geometry abgeleitete Klassen die Definition von Geometriedaten:



Je nach Typ, benötigen Geometry Objekte die Koordinaten von Punkten, Normalvektoren, Farbdefinitionen (falls nicht in der Appearance global für das Objekt festgelegt), Texturkoordinaten, Indices für die Zuordnung der Punkte (für Indexed-Typen). In der Version 1.1 von Java3D enthält jedes Shape3D Objekt genau ein Geometry Objekt. Version 1.2 erlaubt die Angabe einer Liste von äquivalenten Objekten – Triangle und Quad-Typen sind äquivalent und können vermischt werden, ebenso Point- bzw. Line-Typen.

- *Light:* Blattknoten. Von Light abgeleitete Klassen definieren Lichtquellen in der Szene: AmbientLight, DirectionalLight, PointLight, SpotLight.
- *Behavior:* Von Behavior abgeleitete Klassen erlauben ein in den Szenegraphen integrierte Behandlung von Events. Bei Eintreten eines Ereignisses, das mit Hilfe von WakeupCondition-Objekten festgelegt werden kann (z.B Kollisionen, “picking” von Objekten, Mausereignisse, Timer, GUI-Events) wird Code ausgeführt, der Einfluß auf den Szenegraphen nehmen kann: z.B. Transformationen verändern, Lichtquellen aktivieren, etc.)

Helper Klassen & Utilities

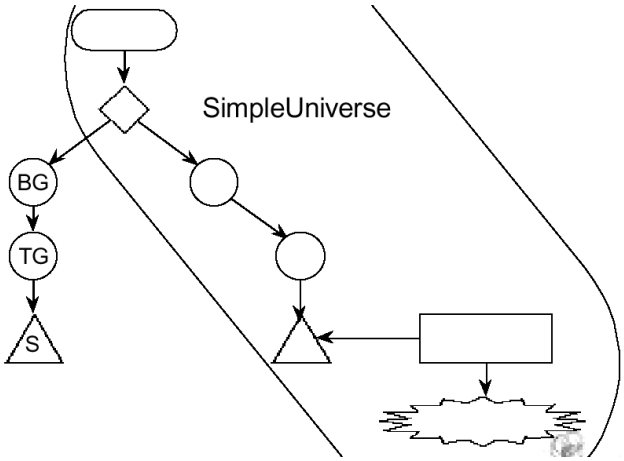
Zusätzlich zu den „Kernklassen“ des Szenegraphen enthält das Java3D API zahlreiche zusätzliche Klassen, die die Programmierung von 3D Applikationen vereinfachen sollen.

- *javax.vecmath.** Das Package vecmath enthält eine Sammlung an Klassen für Vektorberechnungen: Tuple/Point/Vector Klassen in 2,3 und 4D, mit float und double Genauigkeit. Matrix 3/4D und Quaternion Klassen.
- *Javax.media.j3d.Transform3D* Basierend auf einer 4x4 Matrix wird 3D-Graphik spezifische Sichtweise auf eine Matrix implementiert: Translation, Rotationen, Projektionsmatrizen, etc.
- *MouseBehavior* Abgeleitet von der Behavior Klasse wird Manipulation von Transformationen über Mausbewegungen implementiert: Rotation, Skalierung, Translation.
- *SimpleUniverse* Erzeugt den Viewing Teil eines Szenegraphen mit Defaultlichtquelle
- *Simple polygonal objects* Eine Reihe von Klassen kann zur Erzeugung Polygonaler Repräsentationen geometrischer Primitive verwendet werden: Zylinder, Quader, Kugel, Tetraeder, …

Weitere Informationen unter

http://java.sun.com/products/java-media/3D/index.html

Hello3D!




```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import java.appletApplet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;

public class Hello3D extends Applet
{

  public Hello3D()
  {
       setLayout(new BorderLayout());  // auto-layout of components in window
       Canvas3D c= new Canvas3D(null); // component for 3d rendering
       add(„Center“, c);               // place it in center of window
       BranchGroup scene=createSceneGraph(); // make content branch
       SimpleUniverse u=new SimpleUniverse(c);//make simple viewing branch
       u.addBranchGraph(scene);        // add content to scene (make it live)
  }

  public BranchGroup createSceneGraph()// creating the content branch
  {
       BranchGroup objRoot=new BranchGroup();
       Transform3D spin=new Transform3D(); // prepare a transform
       Transform3D tempspin=new Transform3D(); // containing toration around
       spin.rotX(Math.PI/4.0d);                // x and y axis
       tempspin.rotY(Math.PI/5.0d);
       spin.mul(tempspin);
       TransformGroup objTrans=new TransformGroup(spin);
       objRoot.addChild(objTrans);
       objTrans.addChild(new ColorCube());// create geometry using helper object
       return objRoot;
  }

public static void main(String[] args)
  {
   Frame frame=new MainFrame(new Hello3D(), 256, 256);
  }
}
```