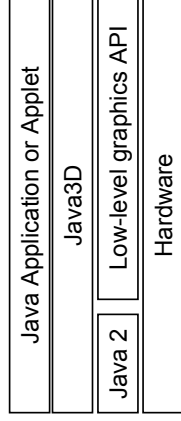


Java3D

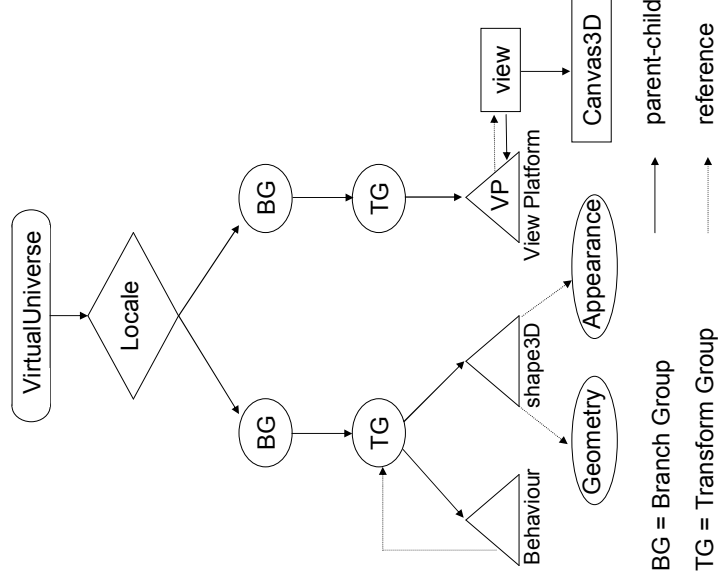
- Java3D is a 3D graphics API with high-level and low-level capabilities.
- It uses a **scene graph** programming model.
 - The scene graph describes the world; the objects in the world, the light sources and the viewpoint.
 - Java3D's runtime and rendering engines figure out how to draw the scene on the display.
- Java3D is layered on top of Java and a low-level graphics API such as OpenGL, Direct3D, etc.



Scene Graph Programming Model

- The scene graph is the basis for Java3D's high-level world modeling.
- A scene graph is a tree-like data structure that stores information about the world.
 - Objects,
 - Appearances,
 - Illuminations, etc.
- A Java3D scene graph consists of **nodes**.
- The top node is the **virtual universe** and its represents the 3D space. This virtual universe is populated with 3D objects.
- The hierarchical properties of a graph allow 3D objects to be grouped and manipulated together.
 - For example, a chair could be modeled as a collection of cuboids. To move the chair we just need to transform the chair node, rather than transforming every cuboid node that makes up the chair.

Example Scene Graph



Scene Graph Objects

- The base class for objects in the scene graph is **SceneGraphObject**.
- The subclasses of **SceneGraphObject** are **Node** and **NodeComponent**.
- The subclasses of **Node** are **Group** and **Leaf**.
- Scene graph objects can be connected by references. A house object could be connected by a reference to an **Appearance** node component. Changing the **Appearance** object to blue will change the house's colour to blue.

Node types

- Nodes are the basic element used to construct a scene graph. They are either:
 - Shape nodes representing 3D objects in the world.
 - Environment nodes that characterise an area of the world, such as the illumination of an area.
 - Group nodes that organise the scene graph.
 - The **ViewPlatform** that defines the place from which the viewer is looking at the world.
- The basic class for shapes is **shape3D**. It consists of **Geometry** and **Appearance** node components.

Group nodes

- The subclasses of a **Group** node are:
 - **BranchGroup** hold subgraphs that can be added or deleted while a scene is being displayed.
 - **Ordergroup** that specifies the order in which its children are rendered.
 - **Switch** that allows the display of each of its children to be turned on and off.
 - **TransformGroup** that allows the transformation of its children (i.e. changing their position, orientation or size).
 - **SharedGroup** that allows a subgraph to be appear more than one group.

Environment nodes

- **Light** nodes define how the scene is illuminated (position, direction, type of light source).
- **Background** nodes define the background for the scene.
- **Fog** nodes define atmospheric effects (fog, smoke). Object in the distance could be faded.
- **Behavior** nodes are usually based on events (mouse click, moving the viewer, time passing) and modify the scene graph.
- **Clip** nodes stop objects far from the viewer being rendered.
- **Sound** nodes define sound sources.
- **Soundscape** nodes modify the listener's sound environment (reverberation, echos, etc.)
- **AlternateAppearance** nodes can be used to override the Appearance components of Shape nodes.

First Java3D program

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
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 javax.vecmath.*;

public class HelloUniverse extends Applet
{
    private SimpleUniverse u = null;

    public BranchGroup createSceneGraph ()
    {
        // Create the root of the branch graph
        BranchGroup objRoot = new BranchGroup ();

        // Create the TransformGroup and initialise to identity.
        // Enable TRANSFORM_WRITE so that our behaviour code
        // can modify it at run time.
        // Add to root of subgraph.
        TransformGroup objTrans = new TransformGroup ();
        objTrans.setCapability
            (TransformGroup.ALLOW_TRANSFORM_WRITE);
        objRoot.addChild (objTrans);
    }
}
```

```

// Create a simple Shape3D node: add it to the scene graph.
objTrans.addChild(new ColorCube(0.4));

//Create a simple Shape3D node and add to scene graph
Transform3D yAxis = new Transform3D ();
Alpha rotationAlpha = new Alpha(-1, 4000);

RotationInterpolator rotator =
new RotationInterpolator
(rotationAlpha, objTrans, yAxis, 0.0f,
(float) Math.PI*2.0f);

BoundingSphere bounds =
new BoundingSphere
(new Point3d (0.0, 0.0, 0.0), 100.0);
rotator.setSchedulingBounds (bounds);
objRoot.addChild (rotator);

// Have Java3D perform optimisation on the scene graph
objRoot.compile ();

return objRoot;
}

```

```

public HelloUniverse () {}

public void init ()
{
    setLayout (new BorderLayout ());
    GraphicsConfiguration config =
SimpleUniverse.getPreferredConfiguration ();
    Canvas3D c = new Canvas3D (config);
    add ("Center", c);

    // Create simple scene and attach it to virtual universe
    BranchGroup scene = createSceneGraph ();
    u = new SimpleUniverse (c);

    // Set ViewPlatform a bit back so objects in scene can be
viewed.
    u.getViewingPlatform ().setNominalViewingTransform();

    u.addBranchGraph (scene);
}

public void destroy ()
{
    u.removeAllLocales ();
}

// Add main so it can be run as an application as well as an
applet
public static void main (String args[])
{
    new MainFrame (new HelloUniverse (), 256, 256);
}
}

```

Points to note

- The structure of a simple Java3D program is:
 - Create a **Canvas3D** to display the scene.
 - Create a **SimpleUniverse** to contain and manage the scene graph.
 - Attach **Canvas3D** to **SimpleUniverse**.
 - Create a content branch to describe the scene to be rendered and connect it to the **SimpleUniverse**.
- **BranchGroups** are the root nodes for branch graphs. Adding a branch graph to a “live” scene make the the branch graph “live” and available for displaying.
 - There are limitations on what you can do to “live” elements.
 - Java3D makes an internal representation of live subgraphs that are optimised for rendering.
 - Detaching a live subgraph loses this work.
 - void moveTo (BranchGroup bg);
 - This method of **Groups** keeps a branch group live.

Capabilities

- In order to efficient render a scene graph, Java3D assumes the elements in the scene do not change. If an element in a scene will change its **capability** bit must be set.
- Capability bits can only be changed when a scene graph is not live.
- Capability bits can be accessed and modified using the methods:
 - void setCapability (int bit);
 - boolean getCapability (int bit);
 - void clearCapability (int bit);
- In the example we need to set ALLOW_TRANSFORM_WRITE to allow the rotator behaviour to modify the cube.
- A **BranchGroup** or **SharedGroup** can be further optimised by calling the **compile()** method. This finalises the capability bit contracts and no further changes to the capability bits are allowed.

Java3D Resources

- Information on Java3D can be found at:
 - www.java3d.dev.java.net
 - Java3D homepage
 - java.sun.com/developer/onlineTraining/java3d
 - Java3D tutorial
 - java3d.j3d.org
 - Java3D community site
 - www.java3d.org
- DCU Library
 - Java 3D : API jump-start
 - The Java 3D API specification