import java.applet.Applet;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.image.TextureLoader;

public class SimpleTest extends Applet {
    public SimpleTest() {
        SimpleUniverse u = new SimpleUniverse();
        u.addBranchGraph(createBackground());

        BranchGroup bg = new BranchGroup();
        TransformGroup tg = createBehaviors(bg);
        tg.addChild(createSceneGraph());
        u.getViewingPlatform().setNominalViewingTransform();
        addLights(bg);
        u.addBranchGraph(bg);
    }
}
public BranchGroup createSceneGraph() {
    Appearance app = new Appearance();
    Color3f color = new Color3f(0.8f, 0.8f, 0.8f);
    Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
    app.setMaterial(new Material(black, black, color, color, 2.0f));

    BranchGroup bg = new BranchGroup();
    bg.addChild(new Sphere(0.1f, Primitive.GENERATE_NORMALS, app));
    return bg;
}

public void addLights(BranchGroup bg) {
    DirectionalLight l = new DirectionalLight(new Color3f(0.4f, 0.4f, 0.4f),
        new Vector3f(0.0f, -1.0f, -1.0f));
    l.setInfluencingBounds(getBoundingSphere());
    bg.addChild(l);
}
public BranchGroup createBackground() {
    Background back = new Background();
    back.setApplicationBounds(getBoundingSphere());

    Appearance app = new Appearance();
    app.setTexture(new TextureLoader("back.jpg", this).getTexture());

    BranchGroup g = new BranchGroup();
    g.addChild(new Sphere(1.0f, Primitive.GENERATE_TEXTURE_COORDS |
                            Primitive.GENERATE_NORMALS_INWARD, app));

    back.setGeometry(g);
    BranchGroup bg = new BranchGroup();
    bg.addChild(back);
    return bg;
}
public TransformGroup createBehaviors(BranchGroup bg) {
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    Alpha xAlpha = new Alpha(-1, Alpha.DECREASING_ENABLE | Alpha.INCREASING_ENABLE,
                             1000, 1000, 5000, 1000, 1000, 10000, 2000, 4000);

    PositionInterpolator i = new PositionInterpolator(xAlpha, tg,
                                                        new Transform3D(), -0.8f,
                                                        0.8f);
    i.setSchedulingBounds(getBoundingSphere());
    tg.addChild(i);
    bg.addChild(tg);
    return tg;
}

BoundingSphere getBoundingSphere() {
    return new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0);
}

public static void main(String[] args) { new SimpleTest(); }
the scenegraph

- objects all derive from the **Node** class
- encapsulated within the **VirtualUniverse**
- directed acyclic graph (**addChild** method -- one parent)
- tree structure via **Group** class
scenegraph classes

- **Management nodes**
  - primarily from `Group`
  - `Locale`, `BranchGroup`, `TransformGroup`, `ViewPlatform`, `Switch`

- **Geometry nodes**
  - deriving from `Leaf`
  - primitives (e.g. `Cylinder`) actually `Group`
  - `Shape3D`, `Background`, ...

- **Control/influence**
  - behaviors, `Morph`, `Light`, `Sound`,
Scenegraph, con’t.

- **VirtualUniverse** contains at least one **Locale** object
- Two distinct branches of the scenegraph
  - Scene
  - View (contains **ViewPlatform**)
    - Contains scaling/rotation/translation for view
    - Responsible for rendering scene from given view
    - Renders to attached **Canvas3D**
scenegraphs, con’t.

• nodes contain bounds (*BoundingSphere* or *BoundingBox*) of all children of the node
  – useful for visibility checking in rendering, ...
  – correlated with level of detail
  – useful for spatially limiting behaviors
  – picking
  – collision detection
WHAT IS A SCENEGRAPH?

You'll need to reuse the circuit scenegraph branch and introduce a new element for the F1 race track application.

Sample scenegraph for the F1 race track application:

- World
  - Grass
    - Circuit
    - Trees
      - Billboards
        - People
      - F1 cars
      - Bales
      - Light
        - F1 car 1
          - Stabilizer
          - Rear Fin
          - Chassis
          - Wheel
        - F1 car N
          - Spokes
          - Rim
          - Tire
Many 3D applications define a complex scenegraph hierarchy. An example is shown in Figure 4.13, which depicts the human arm—a hierarchical model. The scenegraph for our conceptual human arm model is illustrated in Figure 4.14.
Many 3D applications define a complex scenegraph hierarchy. An example is shown in Figure 4.13, which illustrates the human arm as a hierarchical model.

There are two slightly undesirable implications of the scenegraph as we have designed it. First, each joint moves the coordinate system of its children, which can be inconvenient if the application needs to maintain a consistent global coordinate system. Second, the coordinate system of each joint is relative to its parent, so if the parent moves, the child’s coordinate system is shifted as well.

These issues can be mitigated by using a different approach. Instead of moving the coordinate system of each joint, the scene graph could be designed to maintain a consistent global coordinate system. Additionally, the scene graph could be designed to support relative coordinate systems, allowing each joint to maintain its own coordinate system relative to its parent.

Figure 4.13 The human arm—a hierarchical model
procedural rendering in GL

```cpp
void Bone::render(Joint *joint) {
    float halfLength = 0.5*length;

    joint->rotate();
    translate(0.0, halfLength, 0.0);
    renderBone(this);
    translate(0.0, halfLength, 0.0);
}

void ForeLimb::render(float shoulderHalfWidth) {
    pushmatrix();
    translate(shoulderHalfWidth, 0.0, 0.0);
    humerus->render(shoulderJoint);
    pushmatrix();
    float humerusHalfWidth = 0.5*humerus->width;
    translate(humerusHalfWidth - 0.5*ulna->width, 0.0, 0.0);
    ulna->render(elbow);
    popmatrix();
    translate(-humerusHalfWidth + 0.5*ulna->width, 0.0, 0.0);
    radius->render(elbow);
    translate(humerusHalfWidth - 0.5*radius->width, 0.0, 0.0);
    manus->render(wrist);
    popmatrix();
}
```
Explicitly Representing Topology, Geometry, Pose and Animation (+): DinoMorph: Parametric Modeling of Skeletal Structures

Each level in this hierarchy adds specialization. The italicized class names correspond to abstract classes in Java.

Skeleton components are made up of elements, which are made up of bones, which can then be visualized in a certain vantage point. This visualization can be done interactively or by selecting a specific menu option.
public abstract class Part extends BranchGroup implements DinoMorphI {
    protected Model model;
    protected Assembly parent;
    protected String name;

    public Part(String n, Model m) {
        name = n;
        model = m;
        super.setPickable(true);
    }
}

public abstract class Element extends Part {
    protected Element proximal;
    protected Element distal;
    protected Shape shape;
    protected Group next;
    protected Group shapeAttachment;
    protected Transform3D matrix;

    public Element(String name, Model m) {
        super(name, m);
        next = this;
        shapeAttachment = new Group();
        matrix = new Transform3D();
        shapeAttachment.setPickable(true);
    }
}

public class Bone extends Element implements Observer, DinoMorphI {
    protected BoneSpec spec;
    protected Hull hull;
    protected CenterOfMass com;
    protected SculptedShape overlay;
    protected TransformGroup tg1;
    protected TransformGroup tg2;
    protected Transform3D t;
    protected Vector3f v;
    protected boolean shapeAttached;

    public Bone(String name, Model m) {
        super(name, m);
        m.add(this);
        com = new CenterOfMass();
        tg1 = new TransformGroup();
        tg2 = new TransformGroup();
        t = new Transform3D();
        v = new Vector3f();
        tg1.setPickable(true);
        tg2.setPickable(true);
        tg1.setDeferUpdates(true);
        tg2.setDeferUpdates(true);
        addChild(tg1);
        tg1.addChild(tg2);
        shapeAttachment = tg1;
        next = tg2;
    }
}
public class Forelimb extends Assembly {
    private Bone      humerus;
    private Elbow     elbow;
    public  Bone      radius;
    public  Bone      ulna;
    private Joint2DOF ulnocarpal;
    private Carpus    carpus;
    private Manus     manus;

    public Forelimb(char side, Model m) {
        super("forelimb" + side, m);
        humerus    = new Bone("humerus" + side, m);
        elbow      = new Elbow(side, m);
        radius     = new Bone("radius" + side, m);
        ulna       = new Bone("ulna" + side, m);
        ulnocarpal = new Joint2DOF("ulnocarpal" + side, m);
        manus      = new Manus(side, m);
        carpus     = new Carpus(side, m);
        setParents();
        link();
        buildGraph();
        carpus.linkProximal(ulnocarpal);
        manus.linkProximal(ulnocarpal);
    }

    public void acceptVisitor(BoneV v) {
        v.setOrder(this);
        if (proximalToDistal) {
            v.visit(humerus);
            v.visit(ulna);
            v.visit(radius);
            carpus.acceptVisitor(v);
            manus.acceptVisitor(v);
        } else ...
    }
    public abstract class Assembly extends Part implements DinoMorphI {
        protected Element proximal;
        protected boolean proximalToDistal;

        public Assembly(String name, Model m) {
            super(name, m);
            m.add(this);
            setProximalToDistal();
        }

        public abstract void acceptVisitor(BoneV v);
        public abstract void acceptVisitor(LocatorV v);
        public abstract void acceptVisitor(PartV v);
    }
}
OO versus procedural rendering

• in GL
  – call render() on each ‘object’ to visualize
  – traverse structures with visitor pattern

• in Java 3D
  – render by attaching Shape3D’s to the scenegraph
  – Shape3D’s are rendered according to specified Attributes
  – traverse structures with visitor pattern attaching and detaching shapes
Shape3D

- wrapper around a Geometry and its Appearance
- setter/getter of user data (of type Object) associated with instance of Shape3D
- bounds for collision detection, etc.

```java
java.lang.Object
 | +--javax.media.j3d.SceneGraphObject
 |    | +--javax.media.j3d.Node
 |    |    | +--javax.media.j3d.Leaf
 |    |    |    +--javax.media.j3d.Shape3D
```
• Sphere, Cylinder, Cone, Box
• in com.sun.java.j3d.utils
• extends Group, not Shape3D
• carries own TransformGroup for scaling, positioning, ...
• pass in Appearance
• specify capabilities

java.lang.Object
  |--javax.media.j3d.SceneGraphObject
  |   |--javax.media.j3d.Node
  |       |--javax.media.j3d.Group
  |           |--com.sun.j3d.utils.geometry.Primitive
Appearance

• wrapper around aspects of a Shape3D’s appearance
  – LineAttributes, (and Point-, Polygon-, Rendering-, and TextureAttributes)
  – Material (ambient, diffuse, emissive, specular, shininess) ... else use ColoringAttributes
  – texgen, texure
  – transparency

• set capabilities, one at a time (e.g. ALLOW_X_READ)

```
java.lang.Object
  |-- javax.media.j3d.SceneGraphObject
    |-- javax.media.j3d.NodeComponent
      |-- javax.media.j3d.Appearance
```
Attributes

- **PointAttributes** (extends NodeComponent)
  - antialiasing
  - size (in pixels, spherical with antialiasing on)

```java
PointArray p = new PointArray(n,
  geometryArray.COORDINATES |
  geometryArray.COLOR_3);

p.setCoordinate(i, new Point3f(...)); // for all i points
p.setColor(i, new Color3f(...)); // for all i points

Appearance a = new Appearance();
a.setPointAttributes(new PointAttributes(size, true);
Shape3d s = new Shape3D(p, a);
```
Attributes

- PolygonAttributes (extends NodeComponent)
  - culling (back, front, none)
  - mode (fill, lines, points)
  - flipping normals
  - offsetting parent Shape3D in z (away from viewer)

```java
PolygonAttributes pa = new PolygonAttributes();
pa.setCapability(PolygonAttributes.ALLOW_CULL_FACE_WRITE);
pa setCullFace(PolygonAttributes.CULL_BACK);
Appearance a = new Appearance();
a.setPolygonAttributes(pa);
Shape3d s = new Shape3D(p, a);
```
Attributes

- TransparencyAttributes (extends NodeComponent)
  - value from 0 (opaque) to 1 (transparent)
  - for whole Shape3D or per vertex (using COLOR_4)
  - expensive
    - renderer must track transparency of every pixel
      (without transparency, pixel color determined by closest, e.g. via Z-buffer)
  - inaccurate and unpredictable
texture mapping

• increasing apparent scene complexity without increasing polygon count
• wrap a texture (from a digitized image) across a surface
• minimize disparity between content of wrapped image and underlying surface morphology
  – stretching, streaking, incongruity
• in Java 3D:

```java
TextureLoader tx = new TextureLoader(texImage, this);
ap.setTexture(tx.getTexture());

TextureAttributes ta = new TextureAttributes();
ta.setTextureMode(TextureAttributes.MODULATE);
ap.setTextureAttributes(ta);
ap.setMaterial(new Material(white, black, white, white, 80.0f));
```
texture mapping (con’t)

- texture mode (for a given TextureAttribute)
  - MODULATE
    - shade surface pixel by incident lights + material properties
    - multiply result by specified texture color
  - DECAL
    - alpha-replaces shading with unshaded texture
    - replaces alpha with texture’s alpha
- TexCoordGeneration (extends NodeComponent)
  - specify type of coordinates to generate (2, 3, or 4)
  - specify how coord’s are to be generated
    - standard method is OBJECT_LINEAR
beachBall = new DynamicProp("Beach Ball", new GeomSphere(0.25f),
   physicsManager.getWorld(), 0.5f);
beachBall.setSurfaceBounce(0.8f);
beachBall.setDisplacement(new Vector3f(2.0f, 4.0f, 4.0f));

Material mat = new Material();
mat.setColorTarget(Material.DIFFUSE);
mat.setAmbientColor(new Color3f(0.8f, 0.8f, 0.8f));
mat.setShininess(8.0f);

TransparencyAttributes ta = new TransparencyAttributes();
ta.setTransparencyMode(TransparencyAttributes.BLENDED);
ta.setTransparency(0.25f);

reflectionAttributes = new TextureAttributes();
reflectionAttributes.setTextureMode(TextureAttributes.MODULATE);

TextureCubeMap beachBallCM = new TextureCubeMap();
beachBallCM.setBoundaryModeS(Texture.CLAMP_TO_EDGE);
beachBallCM.setBoundaryModeT(Texture.CLAMP_TO_EDGE);

TexCoordGeneration reflectionGeneration = new TexCoordGeneration();
reflectionGeneration.setFormat(TexCoordGeneration.TEXTURE_COORDINATE_3);
reflectionGeneration.setGenMode(TexCoordGeneration.REFLECTION_MAP);
TextureLoader tl = TextureLoader.getInstance();
String beachBallPath = "../data/images/BeachBall.gif";
Texture beachBallBaseMap = tl.getMinMapTexture(beachBallPath);
TexCoordGeneration baseGeneration = new TexCoordGeneration();

TextureUnitState[] tus = {
    new TextureUnitState(beachBallCM, reflectionAttributes, reflectionGeneration),
    new TextureUnitState(beachBallBaseMap, null, baseGeneration)
};

Appearance beachBallAppearance = new Appearance();
beachBallAppearance.setMaterial(mat);
beachBallAppearance.setTransparencyAttributes(ta);
beachBallAppearance.setTextureUnitState(tus);

beachBallShape = new SphereShape(0.5f, 20, beachBallAppearance);
beachBallShape.setPickable(true);
beachBallShape.setOwner(beachBall);
beachBallShape.setAppearanceLocked(true);
Behaviors

• **Extends Leaf**
  – no, behaving things are not children of a parent Behavior, analogous to transformed things being children of a parent TransformGroup
  – subclasses include Interpolatorator, MouseBehavior, PickMouseBehavior, LOD, ...
  – abstract class, subclasses shadow initialization and processStimulus methods.
  – scheduling region
    • Behavior is active (can receive stimuli) when a ViewPlatform’s activation volume within Behavior’s scheduling region.
Behaviors (con’t)

• scheduling interval
  – (partial) order of execution for a set of behaviors whose wakeup conditions satisfied at same time
    • lower interval scheduled before higher
    • within-interval scheduled in any order (e.g. those that `WakeupOnElapsedFrames(0)`
  – initialize method evoked when `BranchGroup` added to virtual universe
    • initialize internal state
    • specify first wakeup condition(s), which could change in subsequent wakeups
      – required for `processStimulus` to be executed
Behaviors (con’t)

• processStimulus invoked when an active ViewPlatform’s volume intersects scheduling region and all wakeup criteria satisfied.
  – could cause other Behavior objects to wake up
  – modifications to scene graph not synchronized with rendering, except:
    • all modifications made by a single behavior instance guaranteed to complete in same frame
    • all behaviors that WakeupOnElapsedFrames(0) guaranteed to complete in same frame
    • (but not changes to geometry, texture by reference)
Behaviors (con’t)

• processStimulus must re-call wakeupOn for it to be executed again, else the Behavior will not be re-activated
• processStimulus may query the Enumeration that tells why the Behavior was activated
• processStimulus executed if:
  – Behavior added to scenegraph
  – bounds intersects ViewPlatform’s activation region
  – Behavior is enabled
  – Behavior’s WakeUpCondition is true
  – View.isBehaviorSchedulerRunning() is true
<table>
<thead>
<tr>
<th>Behavior name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billboard</td>
<td>Aligns a TransformGroup relative to the Viewer</td>
</tr>
<tr>
<td>Interpolator</td>
<td>Abstract base class for all Interpolators</td>
</tr>
<tr>
<td>ColorInterpolator</td>
<td>Interpolates diffuse color of a Material</td>
</tr>
<tr>
<td>PathInterpolator</td>
<td>Abstract base class for all PathInterpolators (linear Interpolation)</td>
</tr>
<tr>
<td>PositionPath-</td>
<td>Interpolates the translation of a TransformGroup along a path</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>RotationPath-</td>
<td>Interpolates the rotation of a TransformGroup along a path</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>RotPosPath-</td>
<td>Interpolates the translation and rotation of a TransformGroup along a path</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>RotPosScale-</td>
<td>Interpolates the translation, rotation and scale of a TransformGroup along</td>
</tr>
<tr>
<td>PathInterpolator</td>
<td>a path</td>
</tr>
<tr>
<td>Position-Interpolator</td>
<td>Interpolates the translation of a TransformGroup between two points</td>
</tr>
<tr>
<td>Rotation-Interpolator</td>
<td>Interpolates the rotation of a TransformGroup between two values</td>
</tr>
<tr>
<td>ScaleInterpolator</td>
<td>Interpolates the scale of a TransformGroup between two values</td>
</tr>
<tr>
<td>SwitchValue-</td>
<td>Interpolates between two Switch values, switching on the children of the</td>
</tr>
<tr>
<td>Interpolator</td>
<td>Switch Node</td>
</tr>
<tr>
<td>TCBSplinePath-</td>
<td>Abstract base class for the Spline path Interpolators</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>RotPosScale-TCB</td>
<td>Performs cubic spline interpolation between key frames for the translation,</td>
</tr>
<tr>
<td>SplinePath-</td>
<td>rotation and scale of a TransformGroup</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>Transparency-</td>
<td>Interpolates the transparency of a TransparencyAttribute between two values</td>
</tr>
<tr>
<td>Interpolator</td>
<td></td>
</tr>
<tr>
<td>KeyNavigator-</td>
<td>Simple keyboard navigation by modifying a TransformGroup in response to key</td>
</tr>
<tr>
<td>Behavior</td>
<td>presses</td>
</tr>
<tr>
<td>LOD</td>
<td>Abstract base class for LOD behaviors that modify a Switch Node</td>
</tr>
<tr>
<td>Behavior name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DistanceLOD</td>
<td>Selects child of the Switch Node based on distance from viewer</td>
</tr>
<tr>
<td>MouseBehavior</td>
<td>Abstract base class for the Mouse behaviors</td>
</tr>
<tr>
<td>MouseRotate</td>
<td>Modifies the rotational components of a TransformGroup based on mouse input</td>
</tr>
<tr>
<td>MouseTranslate</td>
<td>Modifies the translation components of a TransformGroup based on mouse input</td>
</tr>
<tr>
<td>MouseZoom</td>
<td>Modifies the scale components of a TransformGroup based on mouse input</td>
</tr>
<tr>
<td>PickMouseBehavior</td>
<td>Abstract base class for the mouse picking behaviors</td>
</tr>
<tr>
<td>PickRotateBehavior</td>
<td>Modifies the rotational components of a picked TransformGroup based on mouse input</td>
</tr>
<tr>
<td>PickTranslateBehavior</td>
<td>Modifies the translation components of a picked TransformGroup based on mouse input</td>
</tr>
<tr>
<td>PickZoomBehavior</td>
<td>Modifies the scale components of a picked TransformGroup based on mouse input</td>
</tr>
</tbody>
</table>
transforming objects

• rotation, translation
• scaling
  – isotropic, anisotropic scaling
  – mirroring by scaling
• compound transformations
  – order of operations matters
X Rotation
Matrix: 
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\theta) & \sin(\theta) & 0 \\
0 & -\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Y Rotation
Matrix: 
\[
\begin{bmatrix}
\cos(\theta) & 0 & -\sin(\theta) & 0 \\
0 & 1 & 0 & 0 \\
\sin(\theta) & 0 & \cos(\theta) & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Z Rotation
Matrix: 
\[
\begin{bmatrix}
\cos(\theta) & \sin(\theta) & 0 & 0 \\
-\sin(\theta) & \cos(\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Scaling
Matrix: 
\[
\begin{bmatrix}
xf & 0 & 0 & 0 \\
0 & yf & 0 & 0 \\
0 & 0 & zf & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Translation
Matrix: 
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
dx & dy & dz & 1 \\
\end{bmatrix}
\]
in Java 3D

• **Transform3D**
  – utility class for holding a Matrix4f transform
  – setters for rotations about x, y, and z
  – setters for translation and scale

• **TransformGroup**
  – extends Group
  – set by passing Transform3D (by value)
translating an object

• hang the object under a TransformGroup
  TransformGroup tg;
  Shape3D s;
  tg.addChild(s);

• apply Transform3D to TransformGroup
  Transform3D t = new Transform3D();
  t.setTranslation(new Vector3f(1.0f, 0.0f, 0.0f));
  tg.setTransform(t);
scaling an object

• hang the object under a TransformGroup
  
  TransformGroup tg;
  Shape3D s;
  tg.addChild(s);

• apply Transform3D to TransformGroup
  
  Transform3D t = new Transform3D();
  t.setScale(new Vector3d(-1.0, 2.0, 1.0));
  tg.setTransform(t);
rotating an object

- **hang the object under a TransformGroup**
  ```java
  TransformGroup tg;
  Shape3D s;
  tg.addChild(s);
  ```

- **apply Transform3D to TransformGroup**
  ```java
  Transform3D t = new Transform3D();
  t.rotX(theta); // in radians
  tg.setTransform(t);
  ```
rotating about an arbitrary axis

• for unit vector \([vx, vy, vz]\) to serve as an axis, and a rotation by \(\theta\)

\[
\text{Vector3f } v = \text{new Vector3f (vx, vy, vz);}
\]
\[
t.\text{set(new AxisAngle4f(v, theta));}
\]

• for example \(t.\text{rotX(\theta)}\) equivalent to:

\[
\text{Vector3f } v = \text{new Vector3f(1.0f, 0.0f, 0.0f);}
\]
\[
t.\text{set(new AxisAngle4f(v, theta));}
\]
combining rotation and translation

- translate then rotate

  Transform3f rotation, translation;
  // initialize rotation, translation matrices
  translation.mul(rotation);
  tg.set(translation);

- result: object rotates about its center then displaced
combining rotation and translation

- rotate then translate
  
  ```java
  Transform3f rotation, translation;
  // initialize rotation, translation matrices
  rotation.mul(translation);
  tg.set(rotation);
  ```

- result: object is rotated also the translation vector is subject to that same rotation
  - also occurs if setting rotation and translation in same Transform3D