CS324e - Elements of Graphics and Visualization

Animation in Java3D

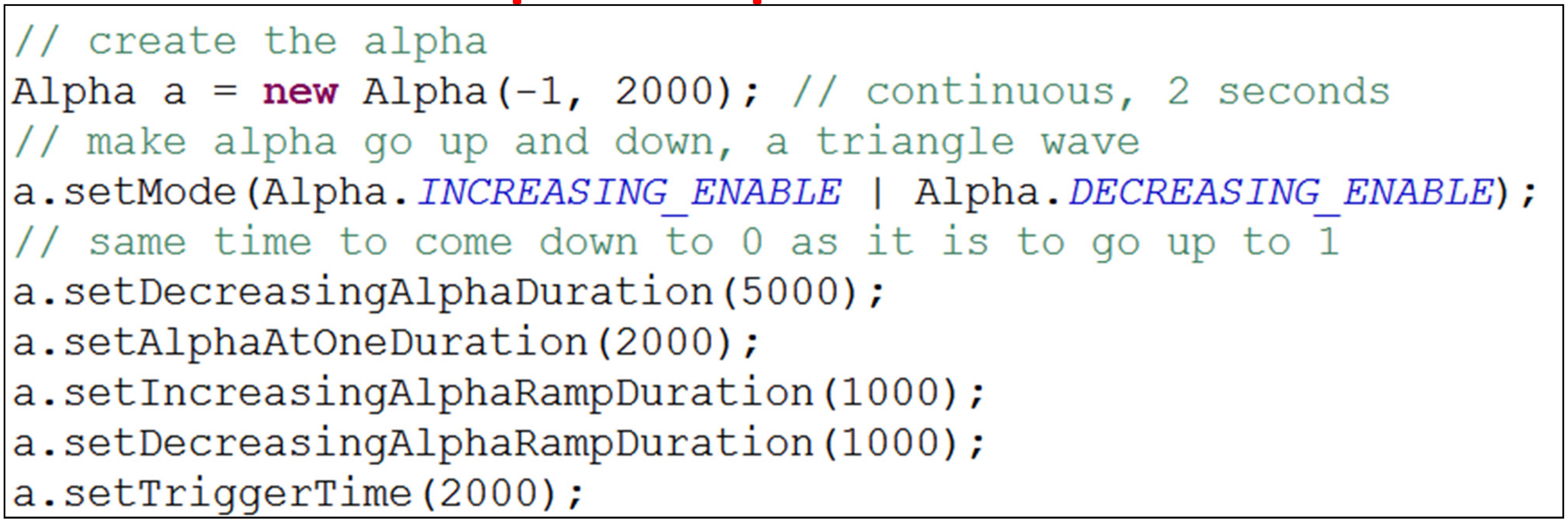
Adding Animation

- Animation in Java2D achieved by changing the position (or some other attribute) of the graphic primitive over time
- Animation framework from FRC simplified things, but still changing attributes of the primitives ourselves
- Different in Java3D

Alphas

- First step to create animation in Java3D is to create an Alpha, Java3D class
  - not to be confused with the alpha channel present in some color spaces
- Similar to the Animator objects from FRC Timing framework
- Generate a series of values over time

Sample Alpha Creation



```
// create the alpha
Alpha a = new Alpha(-1, 2000); // continuous, 2 seconds
// make alpha go up and down, a triangle wave
a.setMode(Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE);
// same time to come down to 0 as it is to go up to 1
a.setDecreasingAlphaDuration(5000);
a.setAlphaAtOneDuration(2000);
a.setIncreasingAlphaRampDuration(1000);
a.setDecreasingAlphaRampDuration(1000);
a.setTriggerTime(2000);
```

– parameters for

– loop count (number of repetitions),

– mode (increasing or decreasing or both)

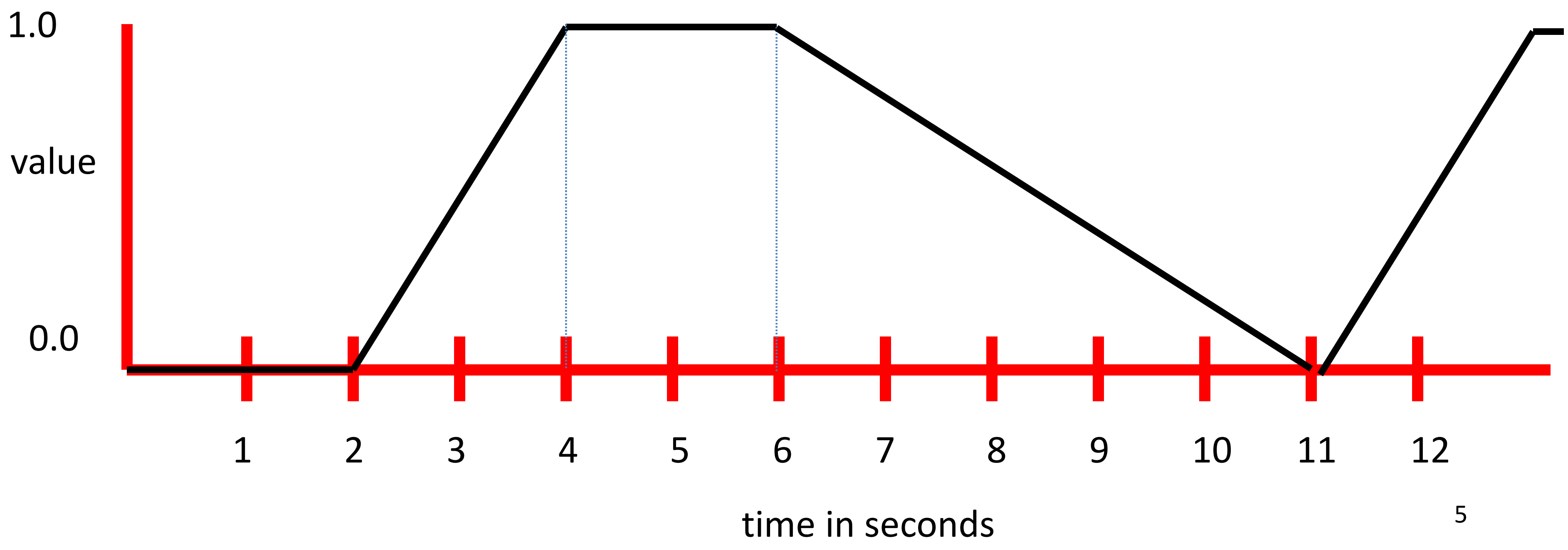
– trigger time, when to start

– duration and ramp times (ramp = acceleration)

– time pause at one and zero

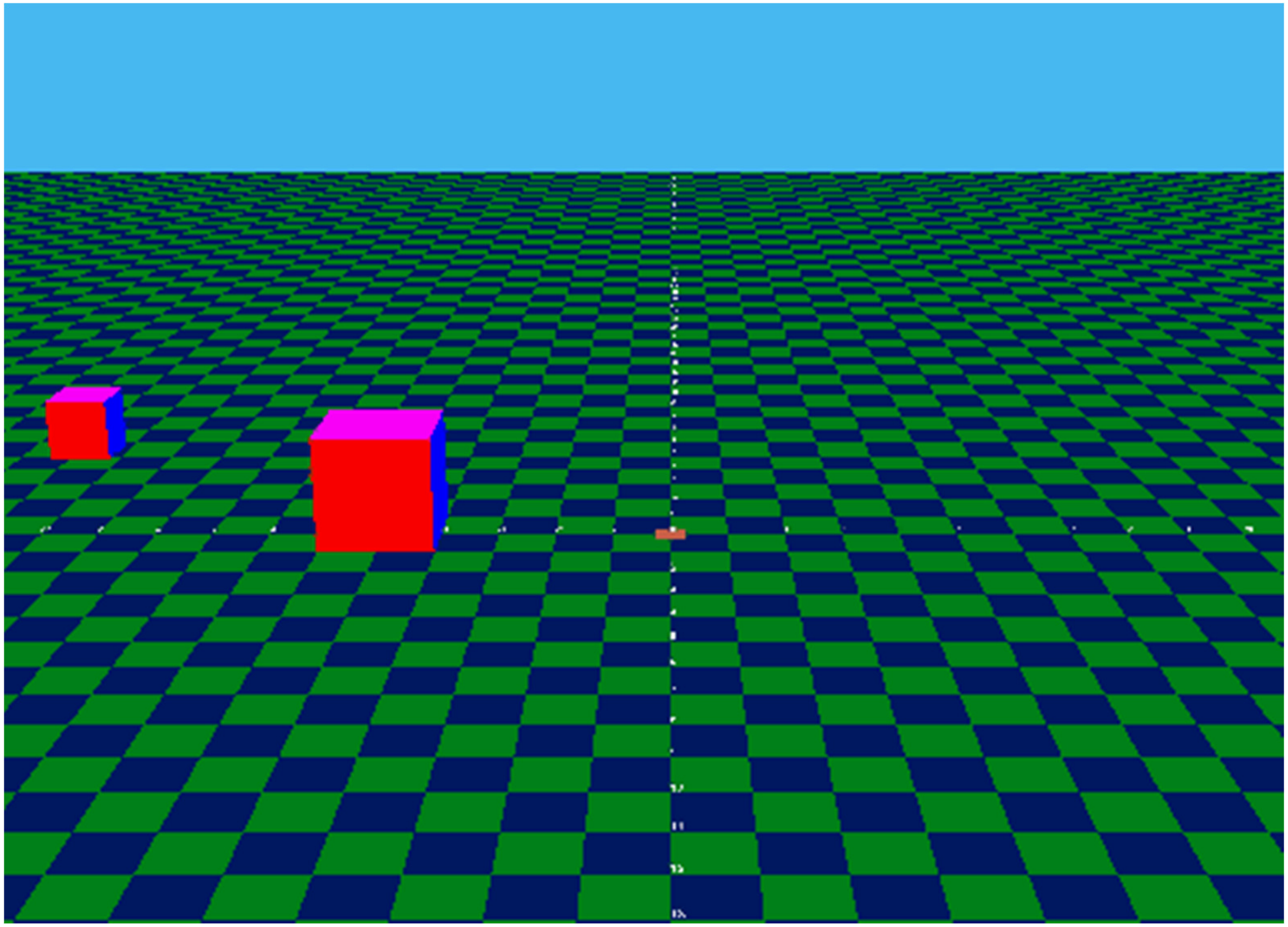
Output of Alpha

- Ignoring the ramp times:



Result

- Notice delay at start and time paused at top

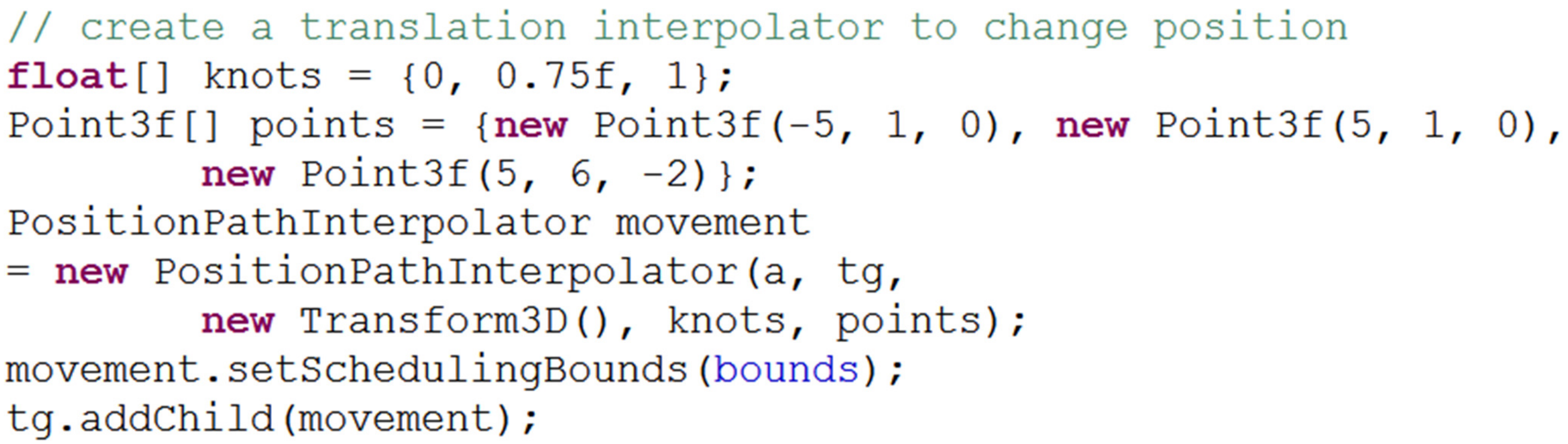


Interpolators

- Alphas are not enough to create animation in the Java3D scene
- results of Alpha must be fed to an *Interpolator* which in turn alters properties of a 3D primitive
- Several kinds of interpolators
- Color, Transform, Transparency, PositionPath, Rotation, Scale,

PositionPath Interpolator

- Movement shown in clip achieved via a PositionPathInterpolator



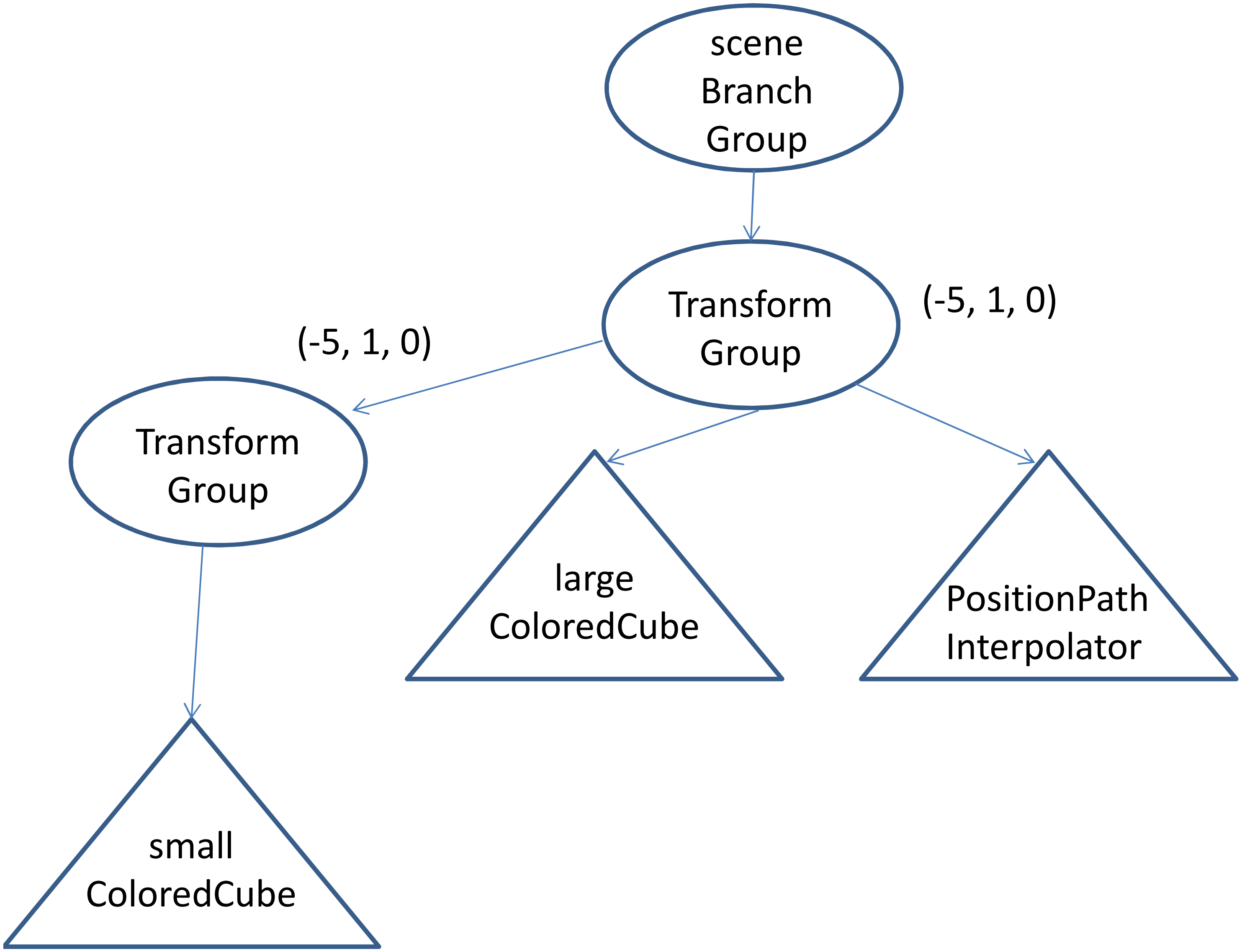
```
// create a translation interpolator to change position
float[] knots = {0, 0.75f, 1};
Point3f[] points = {new Point3f(-5, 1, 0), new Point3f(5, 1, 0),
        new Point3f(5, 6, -2)};
PositionPathInterpolator movement
= new PositionPathInterpolator(a, tg,
        new Transform3D(), knots, points);
movement.setSchedulingBounds(bounds);
tg.addChild(movement);
```

  - a = alpha
  - tg = transform group interpolator acting on
  - Transform3D = local coordinate system
  - knots = how to split up alpha
  - points = positions to move between

Points and Knots

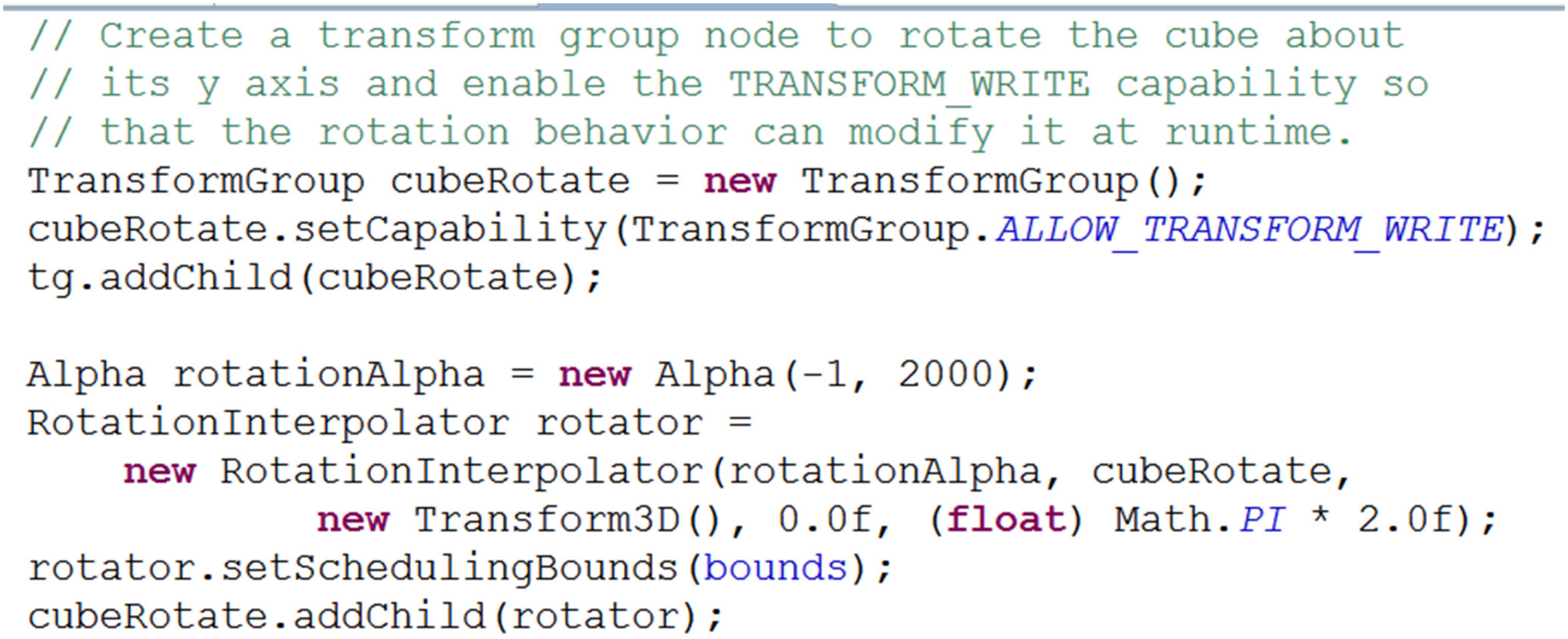
- Points are 3d coordinates
  - start position same as start position of large colored cube
  - what will happen if it isn't?
- Knots are proportions
  - must match number of points or error
  - first must be 0.0, last must be 1.0
  - must be strictly increasing, each value greater than the last
- knots specify what fraction of alpha to move from one point to the next

SceneGraph for Simple Motion



Add Rotation to the Large Cube

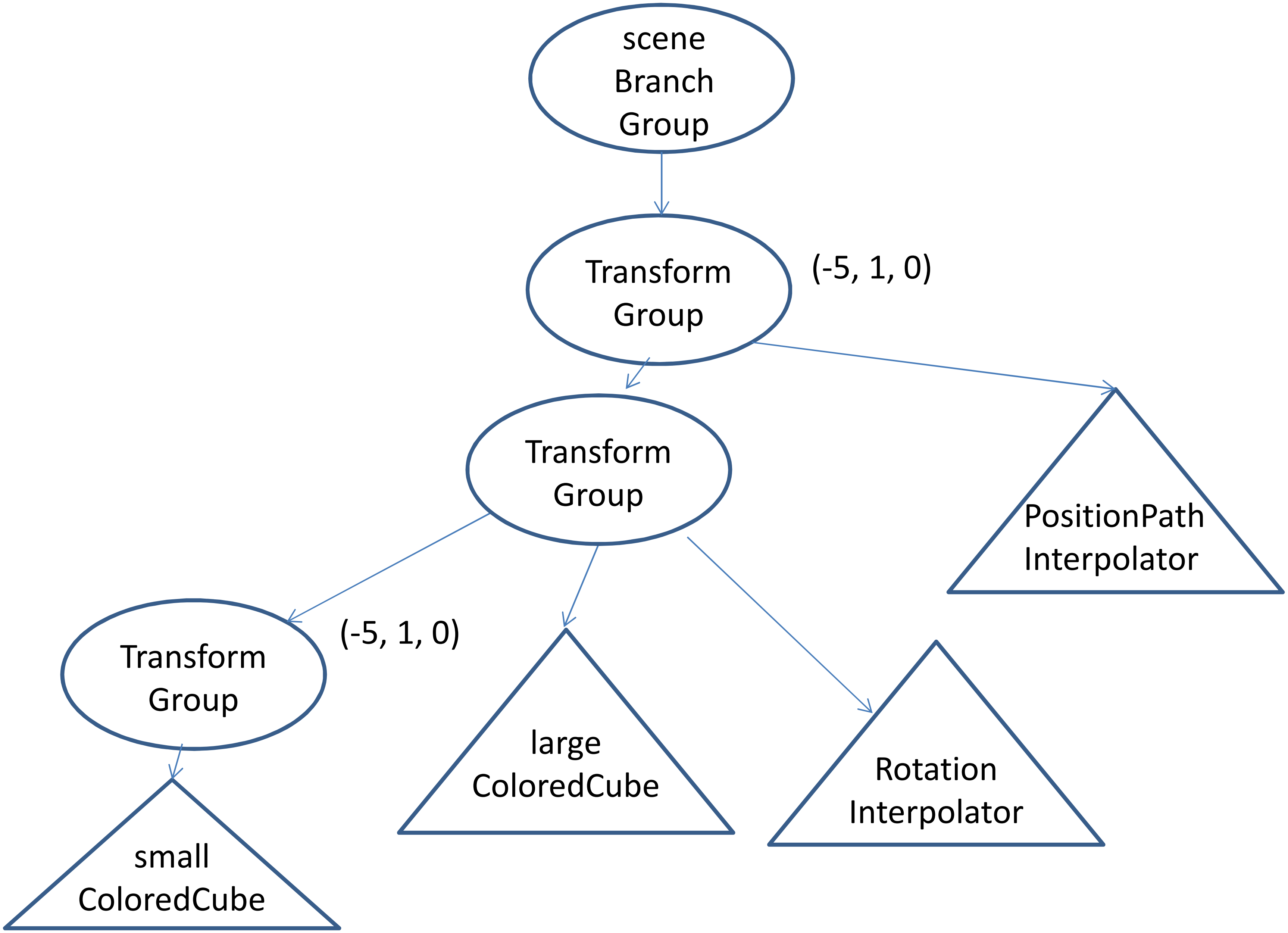
- Another Interpolator



```
// Create a transform group node to rotate the cube about
// its y axis and enable the TRANSFORM_WRITE capability so
// that the rotation behavior can modify it at runtime.
TransformGroup cubeRotate = new TransformGroup();
cubeRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
tg.addChild(cubeRotate);

Alpha rotationAlpha = new Alpha(-1, 2000);
RotationInterpolator rotator =
    new RotationInterpolator(rotationAlpha, cubeRotate,
            new Transform3D(), 0.0f, (float) Math.PI * 2.0f);
rotator.setSchedulingBounds(bounds);
cubeRotate.addChild(rotator);
```

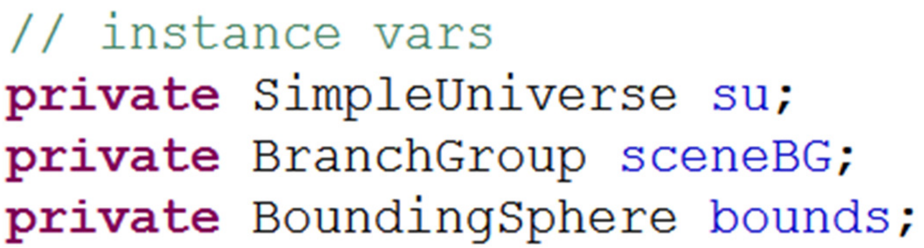
- one approach requires adding another TG which will be rotated

New Scene Graph



Steps to Add Object and Animation

- The panel that holds our 3D scene has a BranchGroup object named sceneBG



```
// instance vars
private SimpleUniverse su;
private BranchGroup sceneBG;
private BoundingSphere bounds;
```

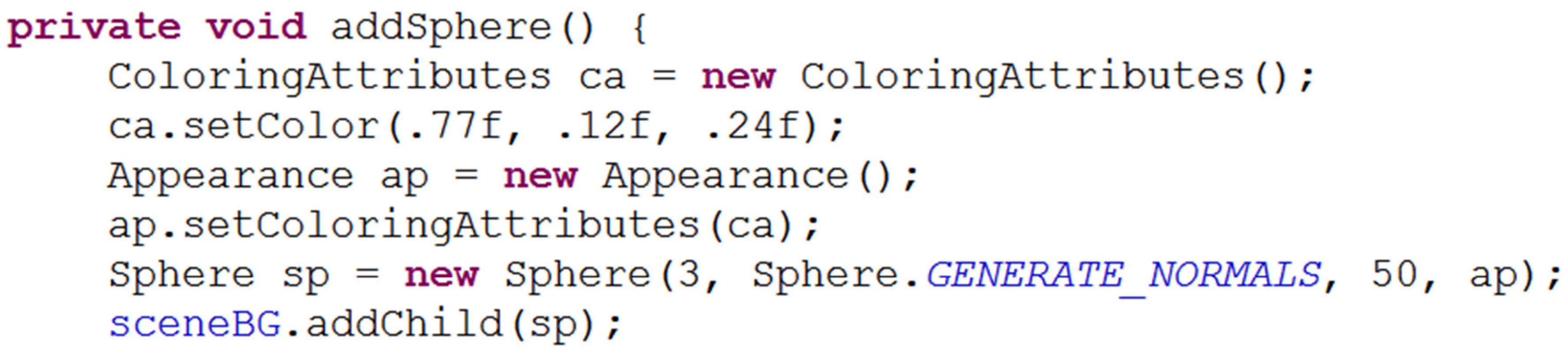
- All of the objects in the scene are children of the sceneBG or descendants

Adding a visible Object to Scene

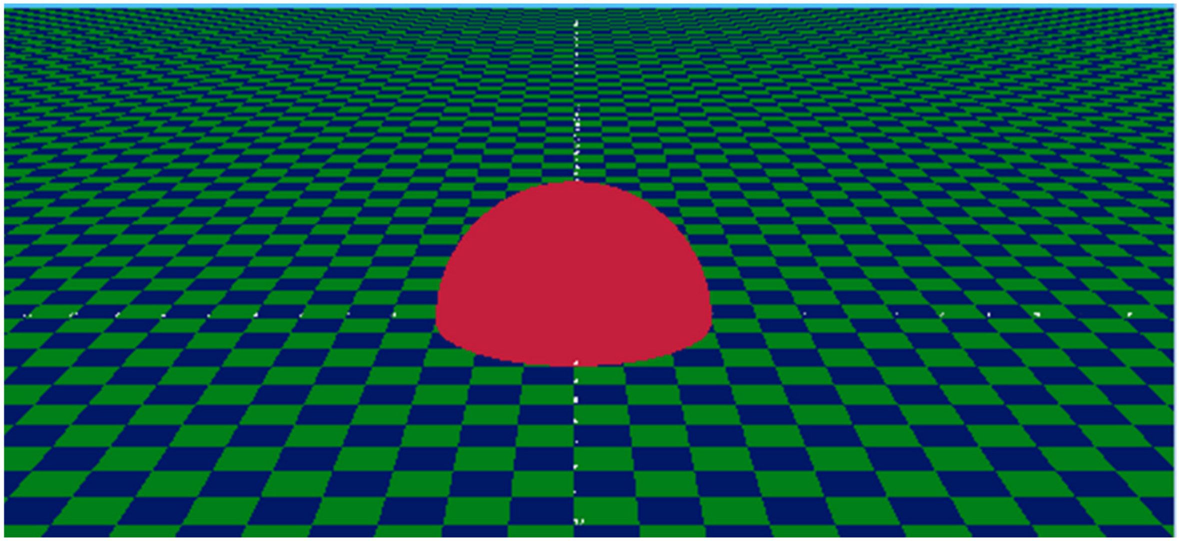
- Add one sphere
- create Appearance for sphere based on ColoringAttributes, Material, or others (line, polygon, texture)
- Create Sphere
  - size, primFlags (GENERATE_NORMALS usually), divisions, appearance

Adding Object to Scene

- If add sphere to sceneBG placed at origin of the scene



```
private void addSphere() {
    ColoringAttributes ca = new ColoringAttributes();
    ca.setColor(.77f, .12f, .24f);
    Appearance ap = new Appearance();
    ap.setColoringAttributes(ca);
    Sphere sp = new Sphere(3, Sphere.GENERATE_NORMALS, 50, ap);
    sceneBG.addChild(sp);
}
```

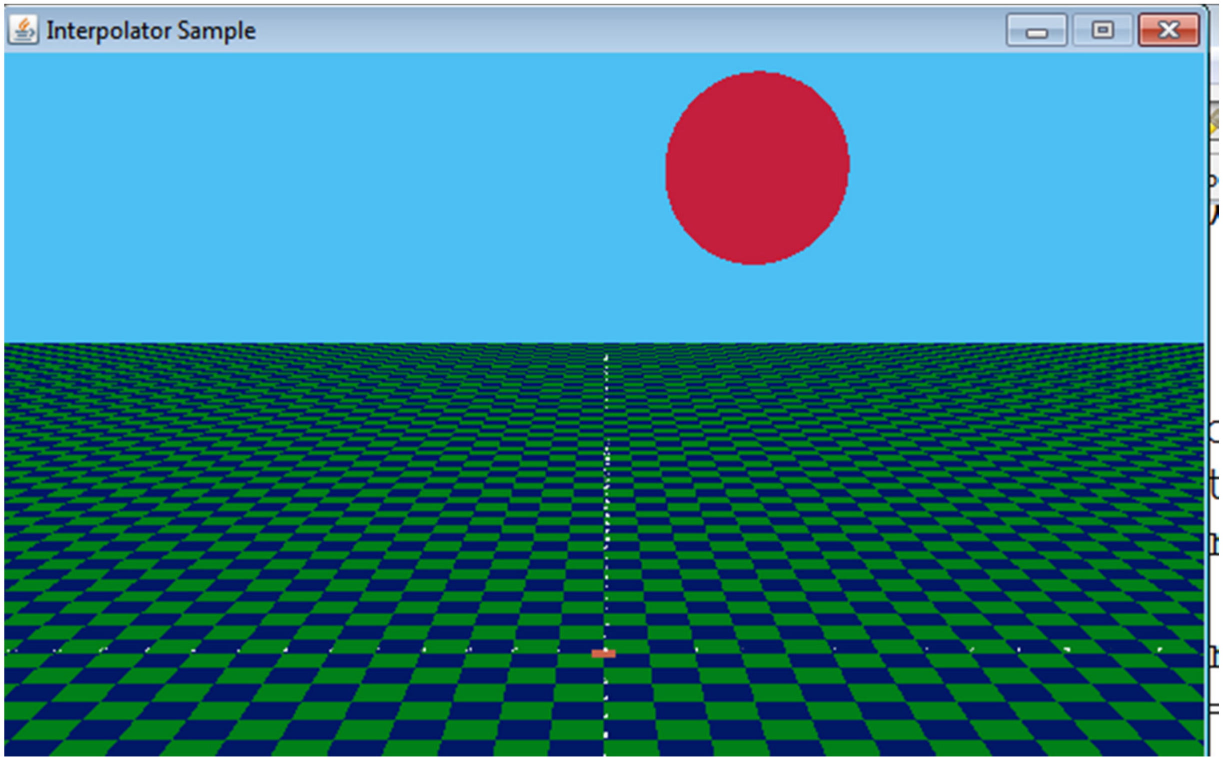


Positioning Sphere

- To position Sphere somewhere besides the origin we must create a Transform3D to specify the position
- Then create a TransformGroup using that Transform3D
- Then add the sphere as a child to that TransformGroup
- Add the transform group as a child to the sceneBG

Positioning Sphere - Code

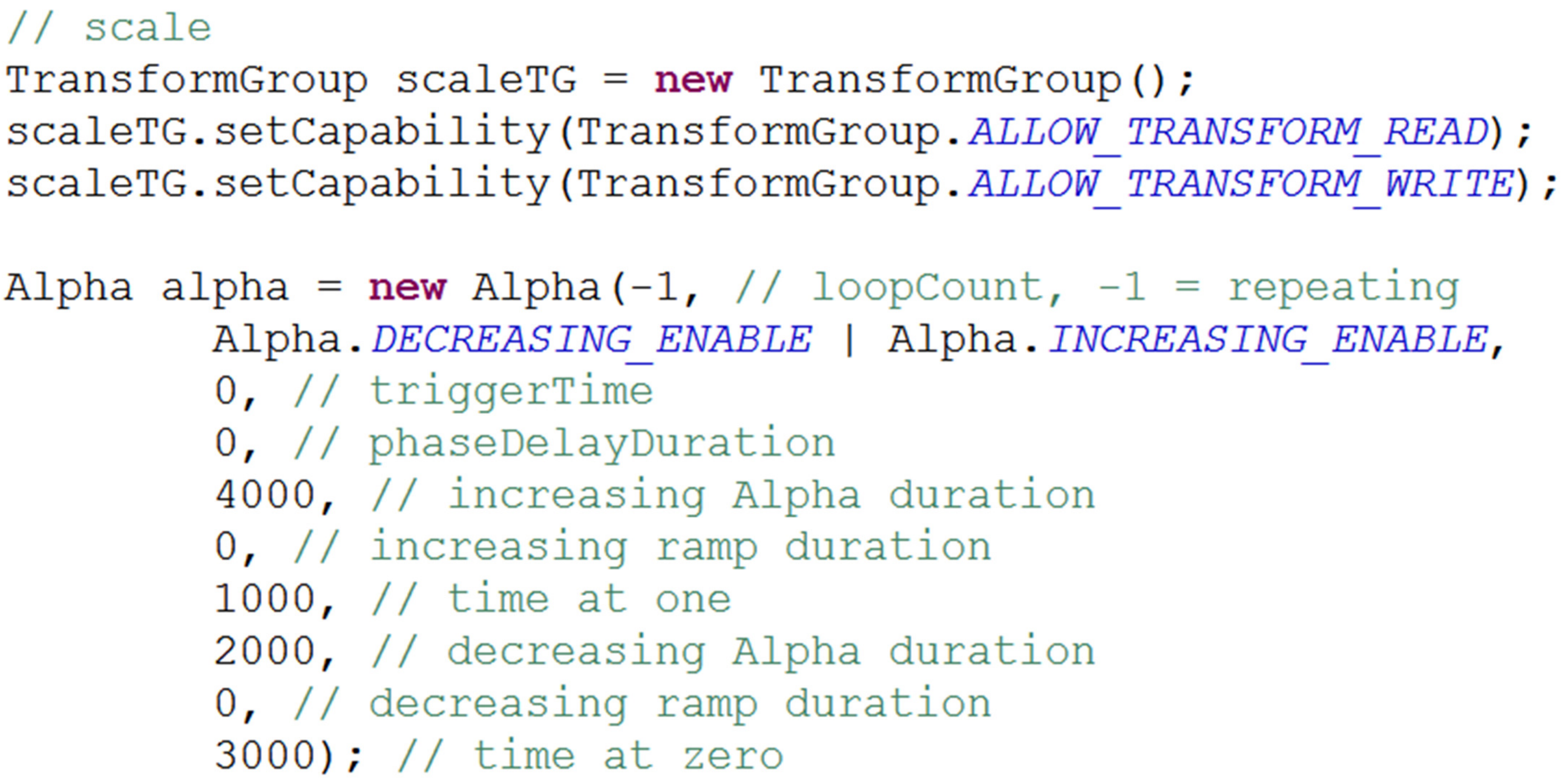
```
    Sphere sp = new Sphere(3, Sphere.GENERATE_NORMALS, 50, ap);

    // position
    Transform3D td3 = new Transform3D();
    td3.setTranslation(new Vector3f(5, 15, -20)); // x, y, z
    TransformGroup tg = new TransformGroup(td3);
    tg.addChild(sp);
    sceneBG.addChild(tg);
}
```



Animate Sphere

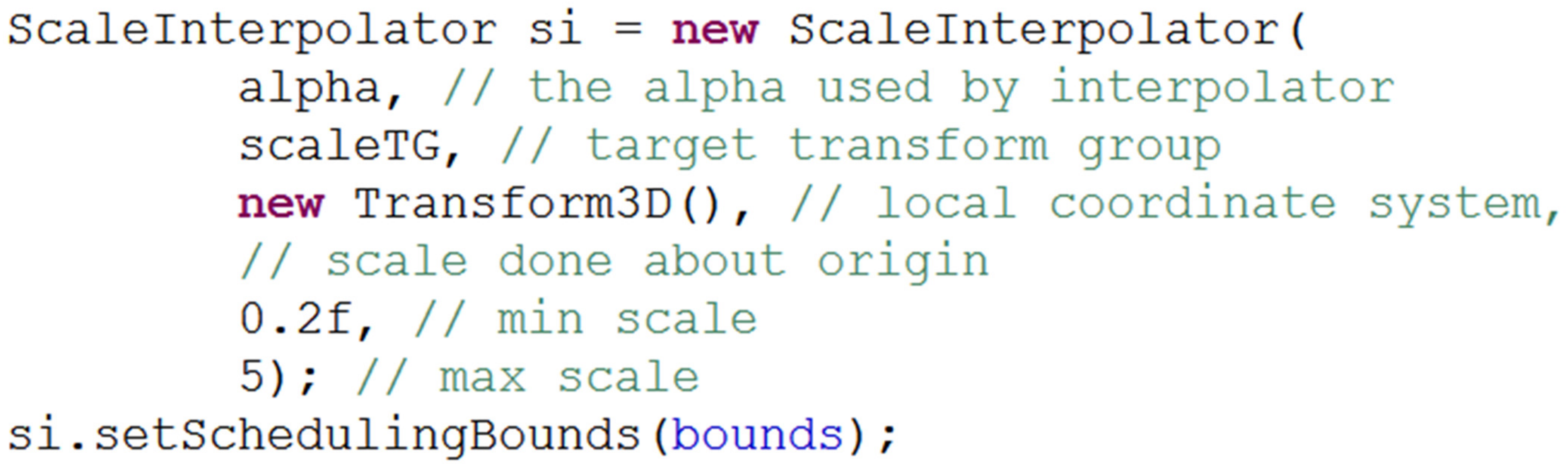
- Create new transform group for interpolator
- set READ and WRITE capabilities on Transform group
- Create Alpha for Interpolator
- Create Interpolator
- Set scheduling bounds for interpolator
- add interpolator to TransformGroup
- add transform group to sceneBG or other transform group
  - to move sphere and have scale change new two transform groups

Create TG for Interpolator and Create Alpha



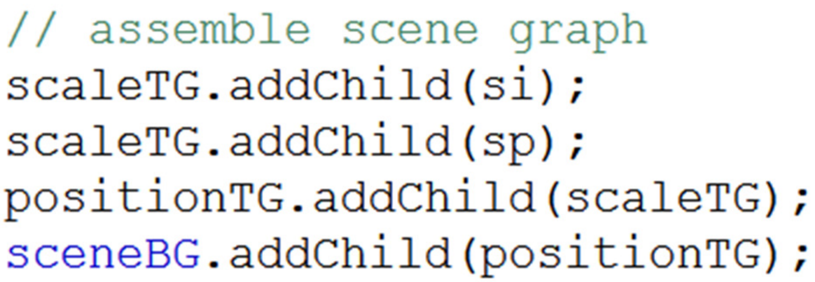
```
// scale
TransformGroup scaleTG = new TransformGroup();
scaleTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
scaleTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

Alpha alpha = new Alpha(-1, // loopCount, -1 = repeating
        Alpha.DECREASING_ENABLE | Alpha.INCREASING_ENABLE,
        0, // triggerTime
        0, // phaseDelayDuration
        4000, // increasing Alpha duration
        0, // increasing ramp duration
        1000, // time at one
        2000, // decreasing Alpha duration
        0, // decreasing ramp duration
        3000); // time at zero
```

Create Interpolator, Set Scheduling Bounds

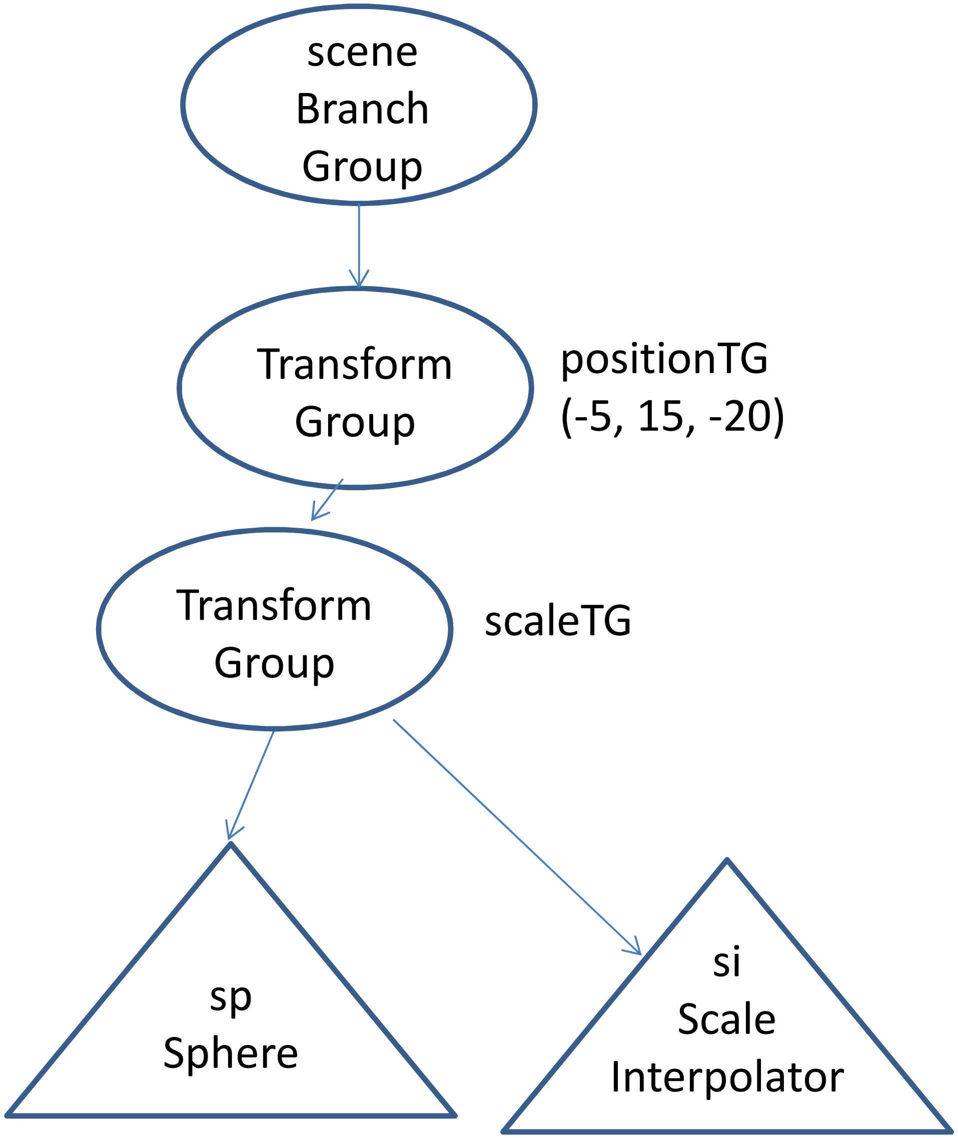


```
ScaleInterpolator si = new ScaleInterpolator(
        alpha, // the alpha used by interpolator
        scaleTG, // target transform group
        new Transform3D(), // local coordinate system,
        // scale done about origin
        0.2f, // min scale
        5); // max scale
si.setSchedulingBounds(bounds);
```

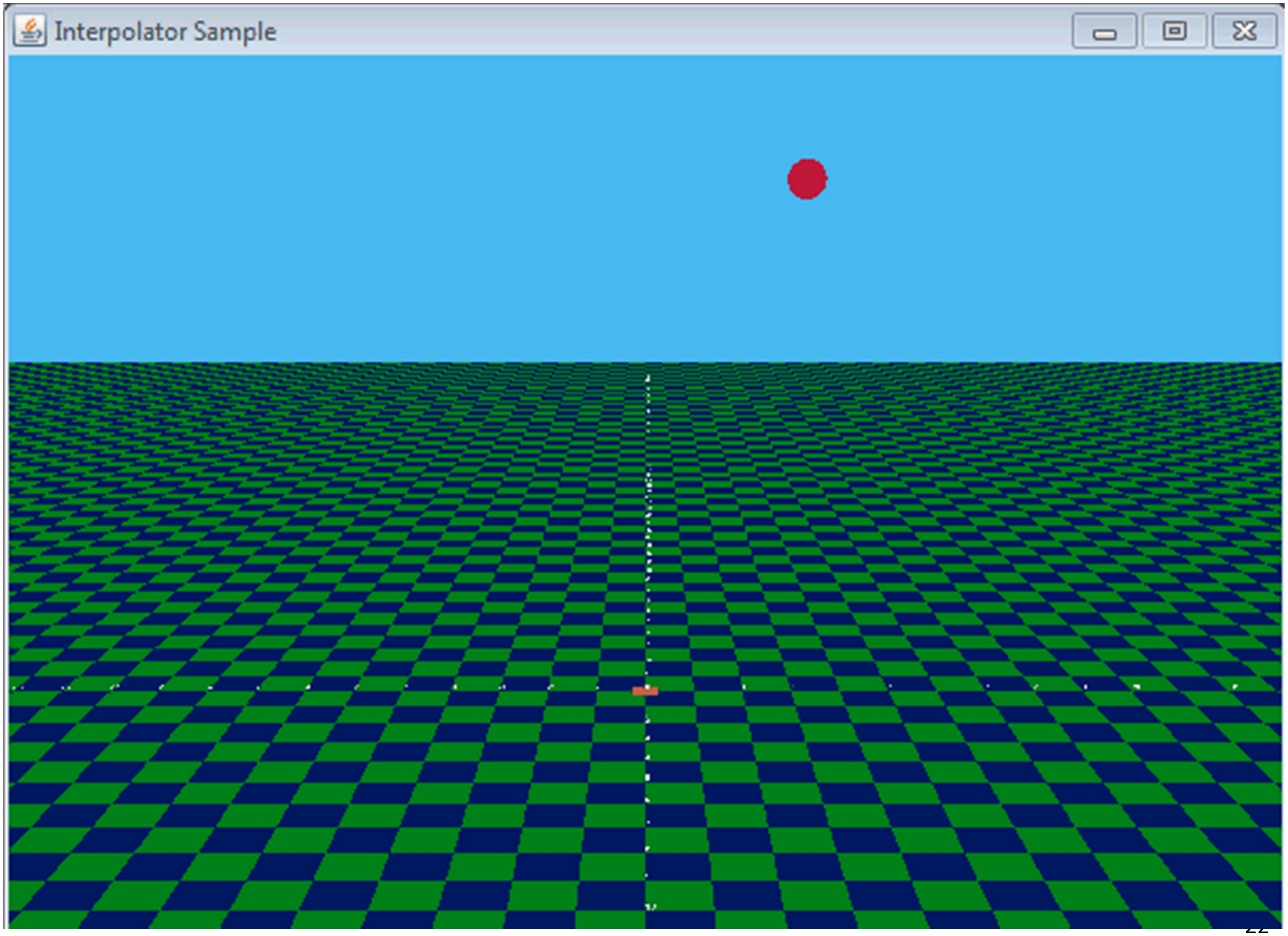
Assemble Scene Graph



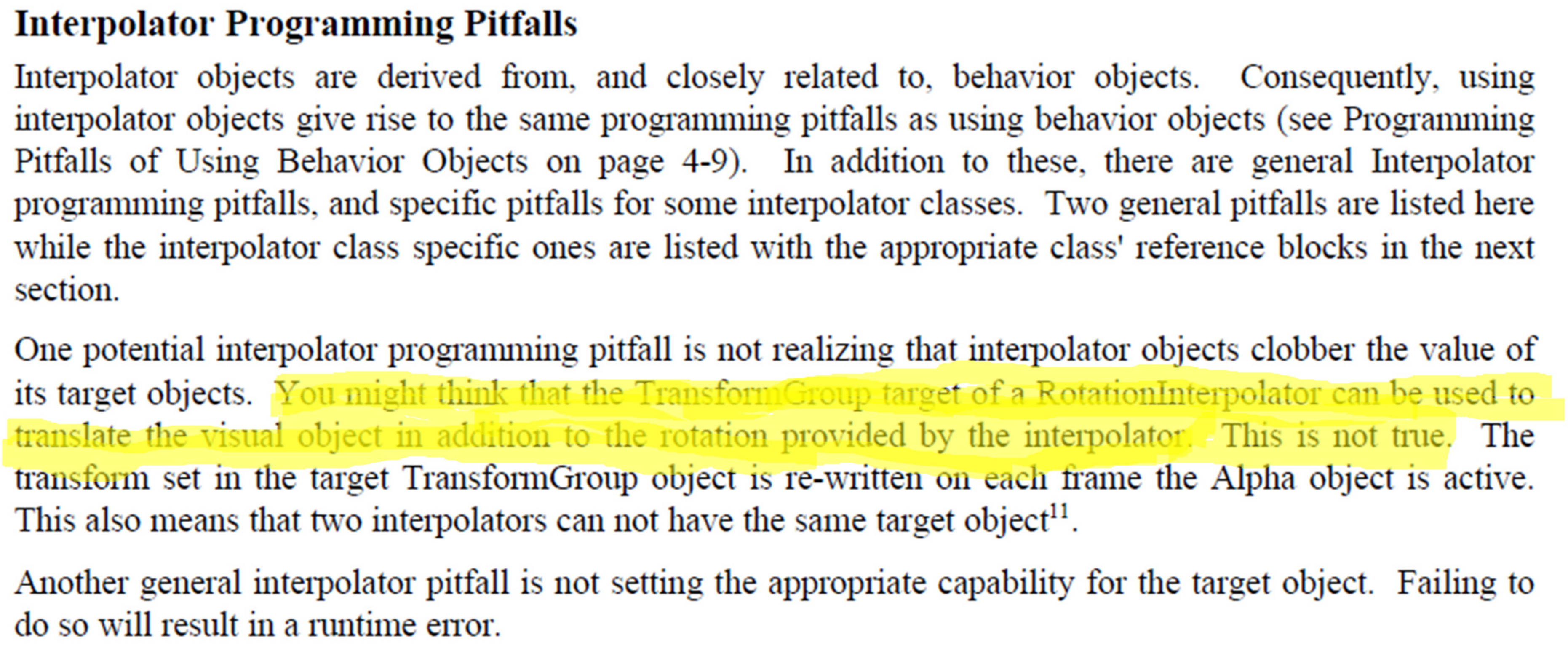
```
// assemble scene graph
scaleTG.addChild(si);
scaleTG.addChild(sp);
positionTG.addChild(scaleTG);
sceneBG.addChild(positionTG);
```



Result



Java3D Tutorial



Interpolator Programming Pitfalls

Interpolator objects are derived from, and closely related to, behavior objects. Consequently, using interpolator objects give rise to the same programming pitfalls as using behavior objects (see Programming Pitfalls of Using Behavior Objects on page 4-9). In addition to these, there are general Interpolator programming pitfalls, and specific pitfalls for some interpolator classes. Two general pitfalls are listed here while the interpolator class specific ones are listed with the appropriate class' reference blocks in the next section.

One potential interpolator programming pitfall is not realizing that interpolator objects clobber the value of its target objects. You might think that the TransformGroup target of a RotationInterpolator can be used to translate the visual object in addition to the rotation provided by the interpolator. This is not true. The transform set in the target TransformGroup object is re-written on each frame the Alpha object is active. This also means that two interpolators can not have the same target object[11].

Another general interpolator pitfall is not setting the appropriate capability for the target object. Failing to do so will result in a runtime error.

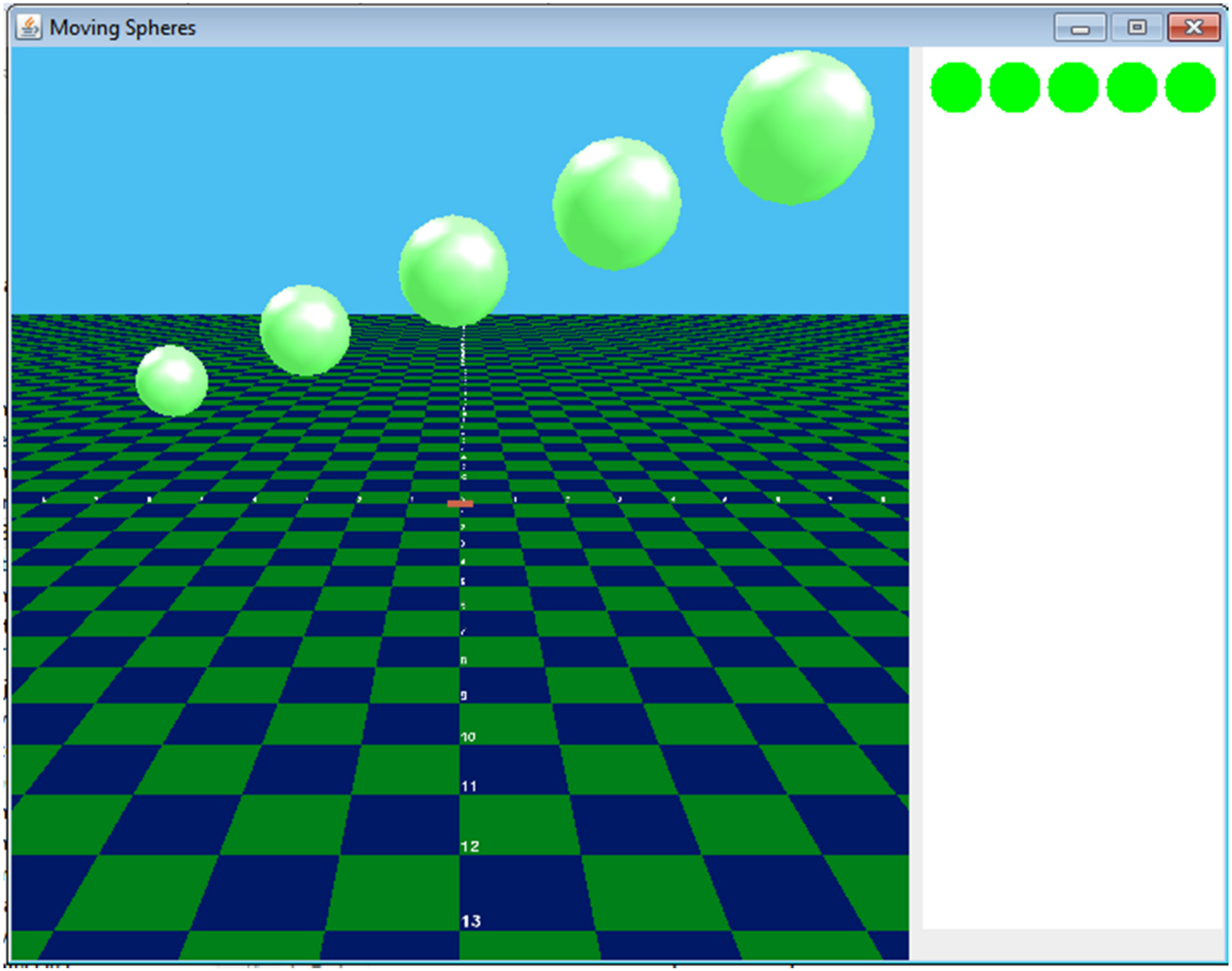
Animation in Java3D:
http://java.sun.com/developer/onlineTraining/java3d/j3d_tutorial_ch5.pdf

General Java3D tutorial:
http://java.sun.com/developer/onlineTraining/java3d/

Controlling Movement

- In the examples so far animation was continuous
- Animation can be controlled via Java3D classes such as *Behaviors* or *Picking*
- Or via a Java2D GUI that causes interpolators to run
- Example of second kind

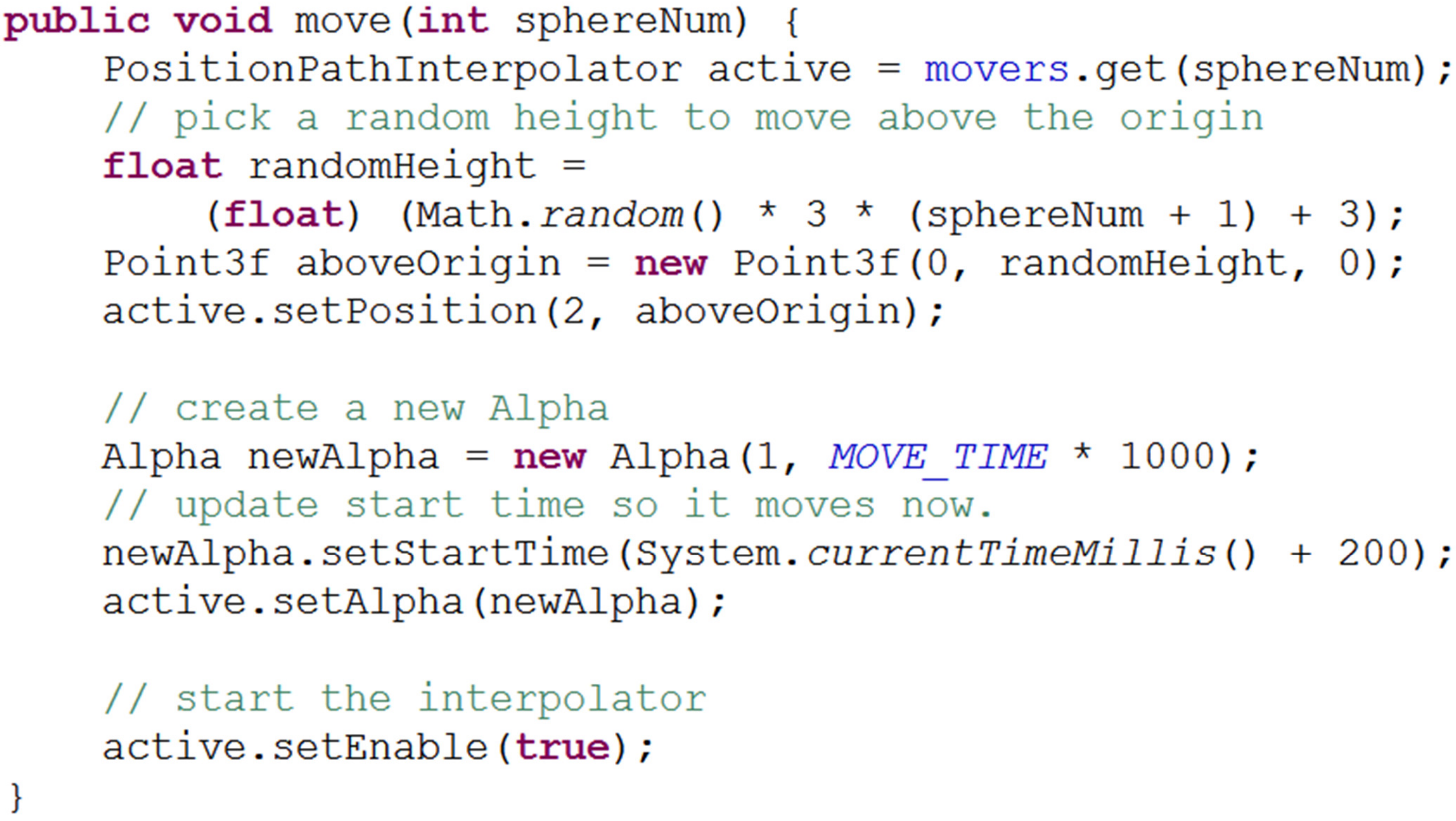
User Initiated Movement



User Activated Movement

- Click on a green circle in panel on right
- creates a new Alpha and starts it
  - move method in 3D world
- Each Sphere part of a graph that has its own PositionPathInterpolator
- Start movement by creating and starting a new Alpha

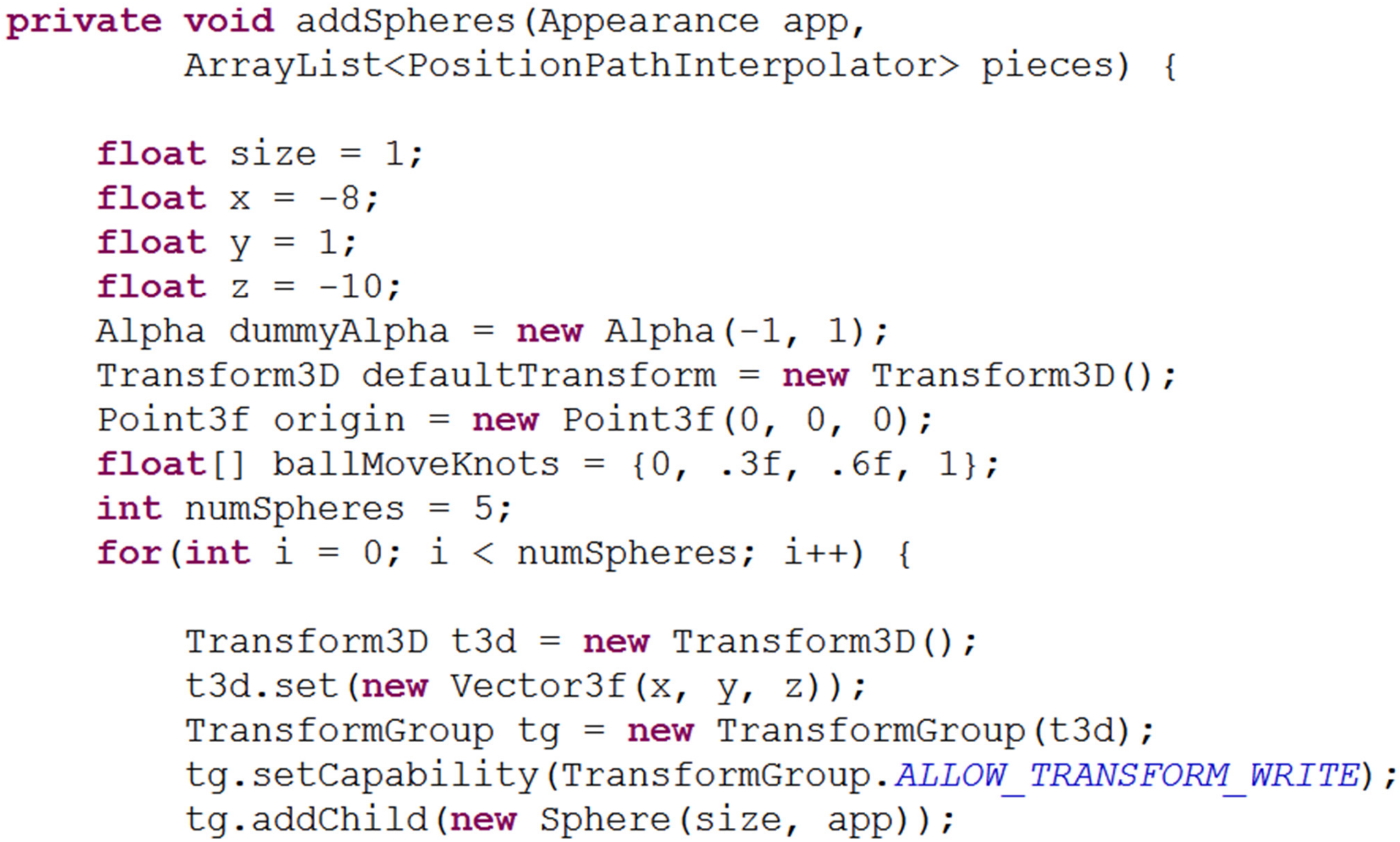
move Method



```
public void move(int sphereNum) {
    PositionPathInterpolator active = movers.get(sphereNum);
    // pick a random height to move above the origin
    float randomHeight =
        (float) (Math.random() * 3 * (sphereNum + 1) + 3);
    Point3f aboveOrigin = new Point3f(0, randomHeight, 0);
    active.setPosition(2, aboveOrigin);

    // create a new Alpha
    Alpha newAlpha = new Alpha(1, MOVE_TIME * 1000);
    // update start time so it moves now.
    newAlpha.setStartTime(System.currentTimeMillis() + 200);
    active.setAlpha(newAlpha);

    // start the interpolator
    active.setEnable(true);
}
```

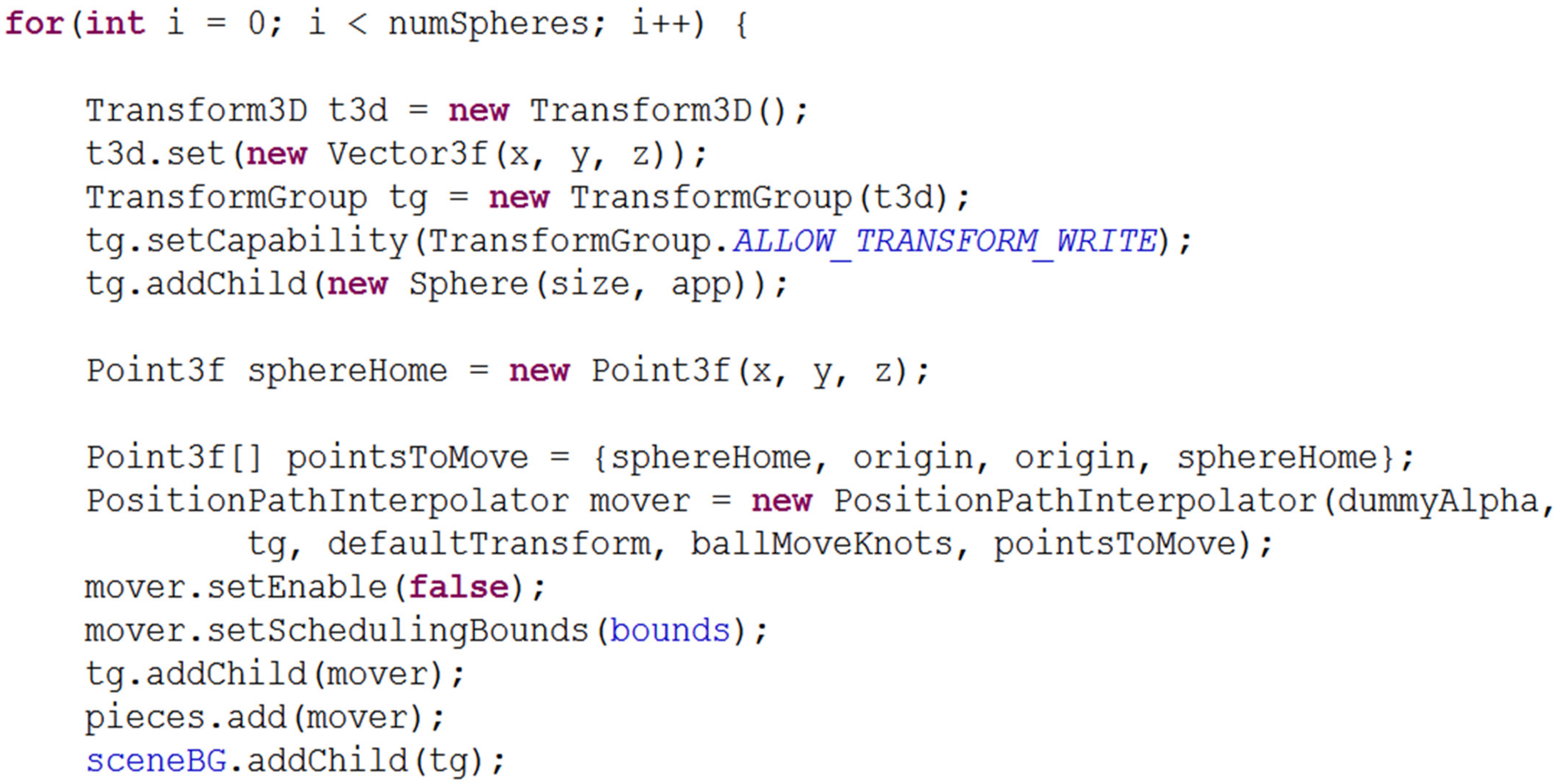
movers

- movers is an ArrayList of PositionPathInterpolators



```java
private void addSpheres(Appearance app,
        ArrayList<PositionPathInterpolator> pieces) {

    float size = 1;
    float x = -8;
    float y = 1;
    float z = -10;
    Alpha dummyAlpha = new Alpha(-1, 1);
    Transform3D defaultTransform = new Transform3D();
    Point3f origin = new Point3f(0, 0, 0);
    float[] ballMoveKnots = {0, .3f, .6f, 1};
    int numSpheres = 5;
    for(int i = 0; i < numSpheres; i++) {

        Transform3D t3d = new Transform3D();
        t3d.set(new Vector3f(x, y, z));
        TransformGroup tg = new TransformGroup(t3d);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        tg.addChild(new Sphere(size, app));
```

Creating and Positioning Spheres



```
for(int i = 0; i < numSpheres; i++) {

    Transform3D t3d = new Transform3D();
    t3d.set(new Vector3f(x, y, z));
    TransformGroup tg = new TransformGroup(t3d);
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg.addChild(new Sphere(size, app));

    Point3f sphereHome = new Point3f(x, y, z);

    Point3f[] pointsToMove = {sphereHome, origin, origin, sphereHome};
    PositionPathInterpolator mover = new PositionPathInterpolator(dummyAlpha,
            tg, defaultTransform, ballMoveKnots, pointsToMove);
    mover.setEnable(false);
    mover.setSchedulingBounds(bounds);
    tg.addChild(mover);
    pieces.add(mover);
    sceneBG.addChild(tg);
```