

Animation and Game Examples Using jME

This document contains several examples which use the jMonkey engine to create and animate objects in a 3D space.

1. **Example 1** *Displaying a box*
2. **Example 2** *Displaying a box and a sphere*
3. **Example 3** *Creating a new light source and using it illuminate the objects in a scene graph*
4. **Example 4** *Animating 3D objects in a scene graph*
5. **Example 5** *Using keyboard events to control the animation of 3D objects*
6. **Example 6** *A Simple Game*

Example1 *Displaying a box on the screen*

This program instantiates a **Box** object and adds it to a scene graph. Upon execution this program will exhibit the following behavior:

- A cube will be displayed in the upper right quadrant of the screen.
- The camera will be positioned in the middle of the screen.
 - Mouse movements are used to rotate the camera at a fixed point in the frustum.
 - The S key moves the camera forward.
 - The W key moves the camera backward.
 - The A key moves the camera to the left.
 - The D key moves the camera to the right.

```
import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.math.Vector3f;

public class Example1 extends SimpleGame{

    public static void main(String[] args) {
        Example1 app = new Example1();
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        app.start();
    }

    protected void simpleInitGame() {
        Box b1;

        Vector3f v1 = new Vector3f(1,1,1);
        Vector3f v2 = new Vector3f(5,5,15);

        b1 = new Box("box", v1, v2);

        rootNode.attachChild(b1);
    }
}
```

The **SimpleGame** class is a simple implementation of the game loop and allows beginning jME programmers to develop simple applications. It will automatically render the scene graph defined in the **simpleInitGame** method. The **simpleInitGame** method is abstract and must be defined whenever **SimpleGame** is extended.

The **Vector3f** objects will be used to hold the x, y and z coordinates for the box.

A **Box** object is being constructed with the following parameters:

- The name is "box"
- The minimum point in the cube is x = 1, y = 1, z = 1
- The maximum point in the cube is x = 5, y = 5, z = 15

The **Box** object is being attached to the root node of the scene graph. Objects must be attached to the scene graph in order to be displayed.

Several three dimensional object classes such as Sphere, Torus, Cylinder, etc..., are available in jME. Example2 modifies the program of Example1 to add a Sphere object to the scene graph.

Example2 *Displaying a box and a sphere on the screen*

This program instantiates a **Box** and a **Sphere** object and adds them both to a scene graph.

```
import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.scene.shape.Sphere;
import com.jme.math.Vector3f;

public class Example2 extends SimpleGame{

    public static void main(String[] args) {
        Example2 app = new Example2();
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        app.start();
    }

    protected void simpleInitGame(){

        Box b1;
        Sphere s1;

        Vector3f v1 = new Vector3f(1,1,1);
        Vector3f v2 = new Vector3f(5,5,15);
        Vector3f v3 = new Vector3f(-3,1,1);

        b1 = new Box("box", v1, v2);
        s1 = new Sphere("sphere", v3, 20, 20, 3);

        rootNode.attachChild(b1);
        rootNode.attachChild(s1);

    }
}
```

A **Sphere** object is being constructed with the following parameters:

- The name is "sphere"
- The center of the sphere is $x = -3, y = 1, z = 1$
- The number of Z samples in the sphere
- The number of radial samples in the sphere

The **Sphere** object is being attached to the root node of the scene graph. The scene graph now contains two three dimensional objects.

Example3 *Creating a new light source and using it illuminate the objects in a scene graph*

This program detaches the default light source from the scene graph, creates a **PointLight** object, gives it the color red and uses it to illuminate the scene graph.

```
import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.scene.shape.Sphere;
import com.jme.renderer.ColorRGBA;
import com.jme.math.Vector3f;
import com.jme.scene.state.LightState;
import com.jme.light.PointLight;

public class Example3 extends SimpleGame{

    public static void main(String[] args) {
        Example3 app = new Example3();
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        app.start();
    }

    protected void simpleInitGame(){

        Box b1;
        Sphere s1;
        PointLight pLight;
        LightState lState;

        Vector3f v1 = new Vector3f(1,1,1);
        Vector3f v2 = new Vector3f(5,5,15);
        Vector3f v3 = new Vector3f(-3,1,1);
        Vector3f v4 = new Vector3f(-5,-4,16);

        b1 = new Box("box", v1, v2);
        s1 = new Sphere("sphere", v3, 20, 20, 3);

        rootNode.attachChild(b1);
        rootNode.attachChild(s1);

        lightState.detachAll();

        pLight = new PointLight();
        pLight.setLocation(v4);
        pLight.setDiffuse(ColorRGBA.red);
        pLight.setEnabled(true);

        lState = display.getRenderer().createLightState();
        lState.attach(pLight);
        rootNode.setRenderState(lState);

    }
}
```

Detach the current light source.

Create a **PointLight** object, give it a location and a color and enable it. **PointLight** objects emit light in every direction.

Create a **LightState** object, attach the **PointLight** object and assign it to the scene graph that it is to illuminate. **LightState** objects are used to modify the color of a scene.

Example4 Animating 3D objects in a scene graph

This program extends the Controller class to create a class which will be used to animate a box a sphere. While we have extended the Controller class to create our own controller in this example, there are several predefined controller classes that can be used to animate objects.

```
import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.scene.shape.Sphere;
import com.jme.renderer.ColorRGBA;
import com.jme.math.Vector3f;
import com.jme.scene.state.LightState;
import com.jme.light.PointLight;
import com.jme.scene.Controller;
import com.jme.scene.TriMesh;

public class Example4 extends SimpleGame
{

    public static void main(String[] args)
    {
        Example4 app = new Example4();
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        app.start();
    }

    protected void simpleInitGame()
    {
        Box b1;
        Sphere s1;
        PointLight pLight;
        LightState lState;

        Vector3f v1 = new Vector3f(5, 6, 4);
        Vector3f v2 = new Vector3f(7, 8, 8);
        Vector3f v3 = new Vector3f(-16, -14, -9);
        Vector3f v4 = new Vector3f(-5, -4, 16);

        b1 = new Box("box", v1, v2);
        s1 = new Sphere("sphere", v3, 20, 20, 1);
        rootNode.attachChild(b1);
        rootNode.attachChild(s1);

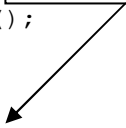
        lightState.detachAll();

        pLight = new PointLight();
        pLight.setLocation(v4);
        pLight.setDiffuse(ColorRGBA.red);
        pLight.setEnabled(true);

        lState = display.getRenderer().createLightState();
        lState.attach(pLight);
        rootNode.setRenderState(lState);

        b1.addController(new ShapeAnimator(b1, -.02f));
        s1.addController(new ShapeAnimator(s1, .02f));
    }
}
```

Create a new ShapeAnimator controller object and attach it to the Box object, b1. Do the same thing for the Sphere object, s1. The ShapeAnimator class is defined later in the program.



```
}
```

```
class ShapeAnimator extends Controller  
{
```

```
    TriMesh shape;  
    Float distance;  
    int count;
```

```
    ShapeAnimator(TriMesh shape, Float distance)
```

```
    {  
        this.shape = shape;  
        this.distance = distance;  
  
        count = 0;  
    }
```

```
    public void update(float time)
```

```
    {  
        Vector3f shapeLoc = shape.getLocalTranslation();
```

```
        if (count > 1500)  
        {  
            distance = distance * -1;  
            count = 0;  
        }
```

```
        shapeLoc.addLocal(distance, distance, distance);  
        shape.setLocalTranslation(shapeLoc);
```

```
        count +=1;  
    }
```

```
}
```

Extend the **Controller** class to create a controller that will be used to translate 3D

The **shape** instance variable constructor will be used to determine which object to translate when the update method is executed and the **distance** instance variable will be used to determine the distance of the translation.

The **update** method overrides an abstract method in class **Controller** to update the object that is being controlled.

Get the current location of the object and place it in a vector.

Add the value in the distance variable to the x, y, and z coordinates in the vector.

Translate the object to the position indicated by the updated x, y, and z coordinates.

Example5 Using keyboard events to control the animation of 3D objects

This program extends the `KeyBoardInputAction` class to create a class which will be used to process keyboard events. When the “K” key is pressed a Sphere object will be created and will be animated to move in a positive direction along the z-axis. When the “L” key is pressed a Box object will be created and will be animated to move in a positive direction along the z-axis. The color of the light illuminating these objects will be changed randomly whenever the “K” or “L” key is pressed.

```
import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.scene.shape.Sphere;
import com.jme.render器.ColorRGBA;
import com.jme.math.Vector3f;
import com.jme.scene.state.LightState;
import com.jme.light.PointLight;
import com.jme.scene.Controller;
import com.jme.scene.TriMesh;
import com.jme.input.KeyInput;
import com.jme.input.action.KeyInputAction;
import com.jme.input.action.InputActionEvent;
```

```
public class Example5 extends SimpleGame
{
```

```
    public static void main(String[] args)
    {
        Example5 app = new Example5();
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        app.start();
    }
```

```
    protected void simpleInitGame()
    {
        lightState.detachAll();
```

```
        input.addKeyboardAction("moveSphere", KeyInput.KEY_K, new MoveShape());
        input.addKeyboardAction("moveBox", KeyInput.KEY_L, new MoveShape());
    }
```

```
class MoveShape extends KeyInputAction
{
    MoveShape()
    {
        setAllowsRepeats(false);
    }
}
```

Assign a particular key to the **KeyInputAction** object that will execute the actions for the key.

The `MoveShape` class extends the abstract class **KeyInputAction** to define actions for keyboard events. Most of the code Example4 to create 3D shape and controller object has been moved into this class so that shapes and controllers will be created when a keyboard event occurs.

The `setAllowsRepeats` method is passed a value of “false” so that keyboard actions will not be executed repetitively when the key associated with them is held down.

```

public void performAction(InputActionEvent e)
{
    Sphere s1;
    Box b1;
    PointLight pLight;
    LightState lState;
    Float distance = .10f;

    if (e.isKeyDown(KeyInput.KEY_K))
    {
        Vector3f v3 = new Vector3f(-1,-1,-20);
        s1 = new Sphere("sphere", v3, 20, 20, 1);
        rootNode.attachChild(s1);
        s1.addController(new ShapeAnimator(s1, distance));
    }
    else if (e.isKeyDown(KeyInput.KEY_L))
    {
        Vector3f v1 = new Vector3f(1,1,-18);
        Vector3f v2 = new Vector3f(2,2,-20);
        b1 = new Box("box", v1, v2);
        rootNode.attachChild(b1);
        b1.addController(new ShapeAnimator(b1, distance));
    }

    Vector3f v4 = new Vector3f(-5,-4,16);
    pLight = new PointLight();
    pLight.setLocation(v4);
    pLight.setDiffuse(ColorRGBA.randomColor());
    pLight.setEnabled(true);

    lState = display.getRenderer().createLightState();
    lState.attach(pLight);
    rootNode.setRenderState(lState);
    rootNode.updateRenderState();
}
}

```

The **performAction** method defines the actions that are to be performed when a keyboard event occurs. The **InputActionEvent** parameter contains an object representing the event that triggered the method call.

When the "K" key is pressed, create a sphere and animate it.

When the "L" key is pressed, create a box and animate it.

```

class ShapeAnimator extends Controller
{
    TriMesh shape;
    Float distance;
    int count;

    ShapeAnimator(TriMesh shape, Float distance)
    {
        this.shape = shape;
        this.distance = distance;
    }

    public void update(float time)
    {
        Vector3f shapeLoc = shape.getLocalTranslation();
        shapeLoc.addLocal(0, 0, distance);
        shape.setLocalTranslation(shapeLoc);
    }
}
}

```

Example6 *A Simple Game*

Tori are placed in 3D space at random positions. A new torus is created every 1000 frames. Once a torus is created it will move back and forth along the z-axis at its given x, y position. Each torus has a ball placed in its center. The ball will oscillate either vertically or horizontally in the center of the torus. The object of the game is to shoot the ball out of the center of the tori by using the mouse to place the crosshair in the correct position and pressing the F-key to fire a missile. When a missile hits a ball the ball will exit the center of the torus along the z-axis and then disappear. When a torus is empty (no longer has a ball in the center) it will rotate and move away from the camera on the z-axis, then disappear.

When a new torus is created the enemy count at the top left of the screen is incremented. When a ball is hit by a missile the point count is incremented by one and the enemy count is decremented by one.

```
import java.util.Random;
import java.util.ArrayList;
import java.util.logging.Level;
import java.net.URL;

import com.jme.app.SimpleGame;
import com.jme.scene.shape.Box;
import com.jme.scene.shape.Sphere;
import com.jme.renderer.ColorRGBA;
import com.jme.math.Vector3f;
import com.jme.scene.state.LightState;
import com.jme.light.PointLight;
import com.jme.scene.Controller;
import com.jme.scene.TriMesh;
import com.jme.input.KeyInput;
import com.jme.input.action.KeyInputAction;
import com.jme.input.action.InputActionEvent;
import com.jme.scene.Text;
import com.jme.math.Quaternion;
import com.jme.math.FastMath;
import com.jme.scene.shape.Torus;
import com.jme.scene.Node;
import com.jme.scene.state.MaterialState;
import com.jme.scene.state.TextureState;
import com.jme.util.TextureManager;
import com.jme.bounding.BoundingSphere;
import com.jme.util.LoggingSystem;
import com.jme.scene.Skybox;
import com.jme.image.Texture;

public class Example6 extends SimpleGame
{
    Sphere ball;
    Torus torus;
    Skybox landscape;
    Text results;
    String resultText;
    Vector3f posVect = new Vector3f(0,0,0);
    int numFrames = 0;
    int enemy = 0;
    int hits = 0;
```

```

Random r = new Random();
MaterialState redMaterial;
MaterialState blueMaterial;
MaterialState purpleMaterial;
ArrayList <Sphere> ballList = new ArrayList<Sphere>();
ArrayList <Torus> torusList = new ArrayList<Torus>();

public static void main(String[] args)
{
    Example6 app = new Example6();
    app.setDialogBehaviour(SimpleGame.NEVER_SHOW_PROPS_DIALOG);
    app.start();
}

/*****
 * Create the basic objects that will be used in the game
 * and add them to the scene graph.
 */
protected void simpleInitGame()
{
    PointLight pLight;
    LightState lState;
    Text crossHair;
    Quaternion matrix = new Quaternion();

    lightState.detachAll();

    /*****
     * Create material states that will be used to set the
     * color of the objects in the scene graph.
     */
    redMaterial = display.getRenderer().createMaterialState();
    redMaterial.setDiffuse(ColorRGBA.red);
    blueMaterial = display.getRenderer().createMaterialState();
    blueMaterial.setDiffuse(ColorRGBA.blue);
    purpleMaterial = display.getRenderer().createMaterialState();
    purpleMaterial.setDiffuse(new ColorRGBA(.626f, .125f, .941f, 0));

    /*****
     * Create a light source set its location and then attach
     * it to the scene graph.
     */
    pLight = new PointLight();
    pLight.setLocation(new Vector3f(40, 40, 10));
    pLight.setDiffuse(ColorRGBA.white);
    pLight.setEnabled(true);
    lState = display.getRenderer().createLightState();
    lState.attach(pLight);
    rootNode.setRenderState(lState);
    rootNode.updateRenderState();

    /*****
     * Designate the F-key keypress event to trigger the creation
     * of a new MissileLauncher contorller object.
     */
    input.addKeyboardAction("launchMissile", KeyInput.KEY_F,
        new MissileLauncher());
}

```

```

/*****
 * Create a text object to hold the crosshair, give it a
 * position relative to the camera, and attach it
 * to the fpsNode. This will cause the crosshair to remain at
 * a fixed position in the middle of the screen.
 */
crossHair = new Text("Launcher", "+");
crossHair.setTextColor(ColorRGBA.green);
crossHair.setLocalTranslation(new Vector3f(display.getWidth() / 2f - 8f,
    display.getHeight() / 2f - 8f, 0));
fpsNode.attachChild(crossHair);

/*****
 * Create a text object to hold the scoring results for
 * the game, give the object a location, and attach it to
 * the fpsNode. This will cause the text to remain fixed in
 * the upper right corner of the screen.
 */
results = new Text("results", "Points: ");
results.setTextColor(ColorRGBA.green);
results.setLocalTranslation(new Vector3f((display.getWidth() / 2f - 8f)
    + -270, (display.getHeight() / 2f - 8f) + 220, 0));
fpsNode.attachChild(results);

/*****
 * Create a skybox object and attach .jpg graphics to each of the panels in
 * the skybox. I have used the skybox .jpg graphics shipped with jME for this
 * example. I suggest that you find some interesting .jpg graphics to make your
 * skybox unique after you test this example.
 */

landscape = new Skybox("landscape", 200, 200, 200);
URL TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/south.jpg");
landscape.setTexture(Skybox.SOUTH, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));
TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/north.jpg");
landscape.setTexture(Skybox.NORTH, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));
TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/east.jpg");
landscape.setTexture(Skybox.EAST, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));
TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/west.jpg");
landscape.setTexture(Skybox.WEST, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));
TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/top.jpg");
landscape.setTexture(Skybox.UP, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));
TexLoc = MyFirstGame.class.getClassLoader().getResource (
    "jmetest/data/texture/bottom.jpg");

```

```

landscape.setTexture(Skybox.DOWN, TextureManager.loadTexture(TexLoc,
    Texture.MM_LINEAR, Texture.FM_LINEAR));

rootNode.attachChild(landscape);
rootNode.setForceView(true);
}

/*****
 * This method is executed every frame and is being used to create
 * a new torus and ball every 1000 frames.
 */
protected void simpleUpdate()
{
    /*****
     * Check to see if 1000 frames that been executed since the
     * last torus was placed in the 3D space. If so, determine the
     * position of the torus and the direction of the ball oscillation,
     * then create the torus and the ball.
     */
    if (numFrames == 0)
    {
        float x, y, z;
        int direction;

        /*****
         * Determine the x,y, and z coordiantes of the torus.
         */
        if (r.nextFloat() > .5f)
            x = r.nextFloat() * 40;
        else
            x = r.nextFloat() * -40;

        if (r.nextFloat() > .5f)
            y = r.nextFloat() * 40;
        else
            y = r.nextFloat() * -40;

        if (r.nextFloat() > .5f)
            z = r.nextFloat() * 15;
        else
            z = r.nextFloat() * -15;

        ball = new Sphere("ball", new Vector3f(0,0,0), 20, 20, .75f);
        ball.setLocalTranslation(new Vector3f(x,y,z));
        ball.setModelBound(new BoundingSphere());
        ball.updateModelBound();
        rootNode.attachChild(ball);

        torus = new Torus("torus", 20, 20, .75f, 2.25f);
        torus.setLocalTranslation(new Vector3f(x,y,z));
        rootNode.attachChild(torus);
    }
}

```

```

        /*****
        * Determine the direction that the ball will oscillate
        * and create new controller objects to handle the animation
        * of the torus and the ball.
        */
        direction = r.nextInt(2) + 1;
        System.out.println("direction : " + direction);
        ball.addController(new BallAnimator(ball, .01f, direction));
        torus.addController(new TorusMover(torus, ball));
        ballList.add(ball);
        torusList.add(torus);
        enemy += 1;
    }
}

/*****
* This method is executed every frame and is being used to create
* update the render state of several objects in the scene graph
* and fpsNode tree.
*/
protected void simpleRender()
{
    if (numFrames == 0)
    {
        int color;

        ball.setRenderState(redMaterial);
        ball.updateRenderState();
        color = r.nextInt(2) + 1;
        if (color == 1)
            torus.setRenderState(purpleMaterial);
        else
            torus.setRenderState(blueMaterial);
        torus.updateRenderState();
    }
    numFrames = (numFrames + 1) % 1000;

    /*****
    * Update the results text to display the points and
    * number of enemies.
    */
    resultText = "Points: " + hits + " Enemy: " + enemy;

    fpsNode.detachChild(results);
    results = new Text("results", resultText);
    results.setTextColor(ColorRGBA.green);
    results.setLocalTranslation(new Vector3f((display.getWidth() / 2f - 8f)
        + -270, (display.getHeight() / 2f - 8f) + 220, 0));
    fpsNode.attachChild(results);
}

```

```

/*****
 * This controller is used to animate the oscillation of a ball
 * inside a torus.
 */
class BallAnimator extends Controller
{
    TriMesh shape;
    Float distance;
    Float count = 0f;
    Float max = .75f;
    int direction;

    BallAnimator(TriMesh shape, Float distance, int direction)
    {
        this.shape = shape;
        this.distance = distance;
        this.direction = direction;
    }

    public void update(float time)
    {
        Vector3f shapeLoc = shape.getLocalTranslation();

        if (direction == 2)
            shapeLoc.addLocal(0, distance, 0);
        else
            shapeLoc.addLocal(distance, 0, 0);

        if (Math.abs(count) > max)
            distance = distance * -1;

        count = count + distance;
    }
}

/*****
 * The MissileLauncher class acts as an event handler and will
 * create an move a missile when the correct key (F-key) is pressed.
 */
class MissileLauncher extends KeyInputAction
{
    int missileCount;

    MissileLauncher()
    {
        setAllowsRepeats(false);
    }

    public void performAction(InputActionEvent e)
    {
        MaterialState missileMaterial =
            display.getRenderer().createMaterialState();
        missileMaterial.setDiffuse(ColorRGBA.green);
    }
}

```

```

/*****
 * Create a new missile. A missile in this example is
 * simply a sphere.
 */
Sphere missile = new Sphere("missile" + missileCount++, 10, 10,
    .25f);
missile.setModelBound(new BoundingSphere());
missile.updateModelBound();

/*****
 * Set the location of the missile so that it will appear to be
 * directly in front of the camera.
 */
missile.setLocalTranslation(new Vector3f(cam.getLocation()));
missile.setRenderState(missileMaterial);
missile.updateGeometricState(0, true);

/*****
 * Create a new MissileMove controller object and use the
 * direction of the camera to determine that direction
 * that the missile will travel.
 */
missile.addController(new MissileMover(missile,
    new Vector3f(cam.getDirection())));

    rootNode.attachChild(missile);
    missile.updateRenderState();
    Vector3f v=cam.getLocation();
}
}

/*****
 * The MissileMover controller handles that animation of a missile.
 */
class MissileMover extends Controller
{
    TriMesh missile;
    TriMesh ball;
    TriMesh torus;
    Vector3f direction;

    float life = 5;

    MissileMover(TriMesh missile, Vector3f direction)
    {
        this.missile = missile;
        this.direction = direction;
        this.direction.normalizeLocal();
    }
}

```

```

public void update(float time)
{
    life = life - time;

    /*****
     * Once the life variable has been decremented to zero
     * remove the missile from the scene graph.
     */
    if (life < 0)
    {
        rootNode.detachChild(missile);
        missile.removeController(this);
        return;
    }

    /*****
     * Determine the location of the missile and then move
     * the missile.
     */
    Vector3f missilePos = missile.getLocalTranslation();
    missilePos.addLocal(direction.mult(1.1f));
    missile.setLocalTranslation(missilePos);

    /*****
     * Test each of the balls in the scene graph to see if they
     * have collided with a missile.
     */
    for (int i = 0; i < ballList.size(); i++)
    {
        ball = (TriMesh) ballList.get(i);
        torus = (TriMesh) torusList.get(i);

        /*****
         * When a missile and a ball collide create a new controller
         * object to move appropriate ball and torus in a new
         * direction.
         */
        if (missile.getWorldBound().intersects(ball.getWorldBound()))
        {
            Controller c = ball.getController(0);
            ball.removeController(c);
            ball.addController(new TargetHitMover(ball, torus));
            c = torus.getController(0);
            torus.removeController(c);
            life = 0;
            hits += 1;
            enemy -= 1;
        }
    }
}
}

```

```

/*****
 * The TargetHitMove controls the animation of a ball after it has
 * been hit by a missile.
 */
class TargetHitMover extends Controller
{
    TriMesh ballTarget;
    TriMesh torus;
    Vector3f direction;

    float life = 2;
    int initialHit = 0;

    TargetHitMover(TriMesh ballTarget, TriMesh torus)
    {
        this.ballTarget = ballTarget;
        this.torus = torus;
    }

    public void update(float time)
    {
        life = life - time;
        if (life < 0)
        {
            rootNode.detachChild(ballTarget);
            ballTarget.removeController(this);

            /*****
             * Create a new controller object to move the torus.
             */
            torus.addController(new TorusHitMover(torus));
            return;
        }

        /*****
         * Change the color of the ball att different stages in
         * its movement.
         */
        if (initialHit == 0)
        {
            ColorRGBA c = new ColorRGBA(1f, .647f, 0, 0);
            MaterialState hitMaterial =
                display.getRenderer().createMaterialState();
            hitMaterial.setDiffuse(c);
            ballTarget.setRenderState(hitMaterial);
            ballTarget.updateRenderState();
        }
        else if (initialHit == 75)
        {
            MaterialState hitMaterial =
                display.getRenderer().createMaterialState();
            ColorRGBA c = new ColorRGBA(1f, .400f, 0, 0);
            hitMaterial.setDiffuse(c);
            ballTarget.setRenderState(hitMaterial);
            ballTarget.updateRenderState();
        }
    }
}

```

```

else if (initialHit == 100)
{
    MaterialState hitMaterial =
        display.getRenderer().createMaterialState();
    hitMaterial.setDiffuse(ColorRGBA.red);
    ballTarget.setRenderState(hitMaterial);
    ballTarget.updateRenderState();
}

initialHit = (initialHit + 1) % 101;

/*****
 * Determine the position of the ball and then move it
 * in the -z direction.
 */
Vector3f ballUpdate = new Vector3f(0, 0, -.1f);
Vector3f ballPos = ballTarget.getLocalTranslation();
ballPos.addLocal(ballUpdate);
}
}

/*****
 * The TorusHitMover controls the animation of a torus after the
 * ball inside it has been hit by a missile.
 */
class TorusHitMover extends Controller
{
    TriMesh torusTarget;

    Vector3f direction;

    Quaternion matrix;
    Float degree;

    float life = 8;

    TorusHitMover(TriMesh torusTarget)
    {
        this.torusTarget = torusTarget;
        matrix = new Quaternion();
        degree = 0f;
    }

    public void update(float time)
    {
        /*****
         * Once the life variable reaches zero remove
         * the torus from the scene graph.
         */
        life = life - time;
        if (life < 0)
        {
            rootNode.detachChild(torusTarget);
            torusTarget.removeController(this);
            return;
        }
    }
}

```

```

        /*****
        * Determine the position of the torus and then move it
        * in the -z direction.
        */
        Vector3f torusUpdate = new Vector3f(0, 0, -.02f);
        Vector3f torusPos = torusTarget.getLocalTranslation();
        torusPos.addLocal(torusUpdate);

        /*****
        * Rotate the torus as it moves.
        */
        degree += .2f;
        matrix.fromAngleAxis(FastMath.DEG_TO_RAD*degree, new
            Vector3f(1,1,0));
        torusTarget.setLocalRotation(matrix);
    }
}

/*****
* This controller is used to animate the movement of a torus.
*/
class TorusMover extends Controller
{
    Torus torusTarget;
    Sphere ballTarget;
    Vector3f direction;
    Quaternion matrix;
    Float step;
    Float max;

    Vector3f torusPos;
    Vector3f ballPos;

    //float life = 8;

    TorusMover(Torus torusTarget, Sphere ballTarget)
    {
        this.torusTarget = torusTarget;
        this.ballTarget = ballTarget;
        matrix = new Quaternion();
        step = -.03f;
        max = 15f;
    }
}

```

```

/*****
 * Move the torus backward and forward along the z-axis.
 */
public void update(float time)
{
    Vector3f torusUpdate = new Vector3f(0, 0, step);
    torusPos = torusTarget.getLocalTranslation();
    torusPos.addLocal(torusUpdate);
    ballPos = ballTarget.getLocalTranslation();
    ballPos.addLocal(torusUpdate);
    if (Math.abs(torusPos.z) > max)
        step = step * -1;
}
}
}

```