Sound in Java 3D

- NB When using a linux box make sure audio is actually enabled first with a command line test such as –

```
auplay file
```

- Java 3D requires an AudioDevice to be selected
  - AudioDevice can be used to specify mono, stereo, headphones, speakers, distances and angles of speakers, etc.
- PhysicalEnvironment can be queried to identify which AudioDevices are available and can then be used to set the particular AudioDevice which is to be used
  - We're currently having problems with this under linux ☹

AudioDevice 3D & 3DL2

- AudioDevice3D
  - Intended that this interface should be implemented by AudioDevice driver developers using a software or hardware sound engine of their choice
- AudioDevice3DL2
  - Extends AudioDevice3D to include reverb and environmental audio parameters that are defined in the MIDI Level 2 Specification
  - Occlusion supported
    - Sounds travelling indirectly to a listener – e.g. round a corner
  - Obstruction supported
    - Sounds muffled by obstructions – e.g through a wall

Sound Nodes

- BackgroundSound
  - Defines an unattenuated, nonspatialised sound source that has no position or direction
  - This type of sound is simply added to the sound mix without modification and is useful for playing a mono or stereo music track or an ambient sound effect
  - Unlike a Background (visual) node, more than one BackgroundSound node can be simultaneously enabled and active
- PointSound
  - Defines a spatially located sound source whose waves radiate uniformly in all directions from a given location in space
  - It specifies a location and a distance-based gain attenuation for different listener positions
  - ConeSound provides a directional extension to PointSound
- SoundScape
  - Defines an application region and an associated aural attribute component object that controls reverberation and atmospheric properties that affect sound source rendering
  - Multiple Soundscape nodes can be included in a scene graph

Scheduling Bounds and Behaviors

- Sound nodes may have scheduling bounds associated with them which specify a region of audibility
  - Sounds are potentially audible when their scheduling bounds intersect with the activation volume of the ViewPlatform
  - `setSchedulingBounds()`
  - `setSchedulingBoundingLeaf()`
- In Java 3D we can create Behaviors for audio
  - Sounds can be triggered to activate upon certain events
  - Note that we can also define Behaviors which activate sounds defined using the basic Java Platform and JMF APIs

Sounds with the Java Platform

- Java 3D provides much more, of course ☺
- The basic Java Platform supports audio via the `AudioSystem` and `MidiSystem` classes
  - Packages `javax.sound.sampled` and `javax.sound.midi`
- A Sun utility is also available via the `AudioPlayer` class
  - Package `sun.audio`
- The Java Media Framework (JMF) provides a higher-level API

Java Sound

Supports

- AIFF, AU, WAV
  `import javax.sound.sampled.*;`
- MIDI (Types 0 & 1) and RMF
  `import javax.sound.midi.*;`

java.sun.com/j2se/1.5.0/docs/guide/sound/programmer_guide/contents.html

```
File soundFile = new File("file.wav");
AudioInputStream soundTrack =
      AudioSystem.getAudioInputStream(soundFile);
AudioFormat soundFormat = soundTrack.getFormat();
SourceDataLine soundLine = null;
DataLine.Info soundInfo = new
  DataLine.Info(SourceDataLine.class, soundFormat);
soundLine = (SourceDataLine)
  AudioSystem.getLine(soundInfo);
soundLine.open(soundFormat);
soundLine.start();
```

Sound with Sun's `AudioPlayer`

- Associate an audio file with an input stream

```
InputStream in = new FileInputStream("file.wav");
```

- Associate the input stream with an audio stream

```
AudioStream soundTrack = new AudioStream(in);
```

- Use `AudioPlayer` class to control playback

```
AudioPlayer.player.start(soundTrack);
AudioPlayer.player.stop(soundTrack);
```

  Etc.

- Ignore warnings about `sun.audio.*` possibly disappearing in the future ☺

Sound with JMF

Supports

- AIFF, AU, AVI, GSM, MIDI, MOV, MPG, MP2, MVR, WAV

  java.sun.com/products/java-media/jmf/2.1.1/guide/index.html

```
Player player = Manager.createPlayer(mediaURL);

 player.realize();    // Initialise the player
 player.prefetch();   // Further initialisation
 player.start();      // Start playing
 player.stop();       // Stop playing
 player.deallocate(); // Free up connections
 player.close();      // Close the player
```