SOUND GENERATION

1. Java 3D provides three types of *sound*s
   - `Sound` class hierarchy

Class Hierarchy

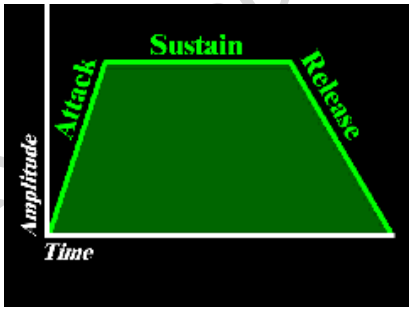
```
java.lang.Object
  └ javax.media.j3d.SceneGraphObject
    └ javax.media.j3d.Node
      └ javax.media.j3d.Leaf
        └ javax.media.j3d.Sound
          ├ javax.media.j3d.BackgroundSound
          └ javax.media.j3d.PointSound
            └ javax.media.j3d.ConeSound
```


   - All three types of sounds have:
     - Sound data to play
     - An initial gain (overall volume)
     - Looping parameters, and playback priority
     - Scheduling bounds (like a behavior)
   - A scene graph can contain multiple sounds.
2. Associated with each `Sound` node is a `MediaContainer`, which includes audio data and information about this data.
   - Typical sound file formats include:
     - AIF: standard cross-platform format
     - AU: standard Sun format
     - WAV: standard PC format

- The sound data can be cached (buffered) or non-cached (unbuffered or streaming).
  - Data can be loaded by a `MediaContainer` from a file on disk or on the Web.
  - If an `AudioDevice` has been attached to the `PhysicalEnvironment`, the sound data is made ready to begin playing.
  - Certain functionality can not be applied to true streaming sound data:
    - Querying the sound's duration (`Sound.DURATION_UNKNOWN` will be returned)
    - Looping over a range of the streaming data, and
    - Restart a previously played portion of the data.

3. Data for non-streaming sound (such as a sound sample) can contain two loop points marking a section of the data that is to be looped specific number of times.

- A sound data can be divided into three segments
  - *Attack*: the start of the sound
  - *Sustain*: the body of the sound
  - *Release*: the ending decay of the sound



- Looping between loop points to sustain a sound
  - Authored using a sound editor
    - They usually bracket the `Sustain` stage
  - If no loop points, loop defaults to entire sound
  - Loops can run a number of times, or forever

4. Controlling sounds
   - Sounds may be enabled and disabled
     - Enabling a sound makes it schedulable
     - The sound will start to play if the sounds scheduling bounds intersect the viewers activation radius
   - Overall sound volume may be controlled with a gain multiplication factor

- By default, sounds are disabled, have a gain of 1.0, and are not looped

5. `BackgroundSound` extends the `Sound` class
   - Similar to `AmbientLight` in lighting, `BackgroundSound` waves come from all directions, flooding an environment at constant volume
   - Use background sounds for:
     - Presentation sounds (voice over, narration)
     - Environment sounds (ocean waves, wind)
     - Background music
   - There could be multiple background sounds playing at the same time

6. BackgroundSound example code

- Load sound data

```
MediaContainer myWave = new MediaContainer("canon.wav");
```

- Create a sound

```
BackgroundSound mySound = new BackgroundSound( );
mySound.setSoundData( myWave );
mySound.setEnable( true );
mySound.setInitialGain( 1.0f );
mySound.setLoop( -1 ); // Loop forever
```

- Set the scheduling bounds

```
BoundingSphere myBounds = new BoundingSphere(
  new Point3d( ), 1000.0 );
mySound.setSchedulingBounds( myBounds );
```

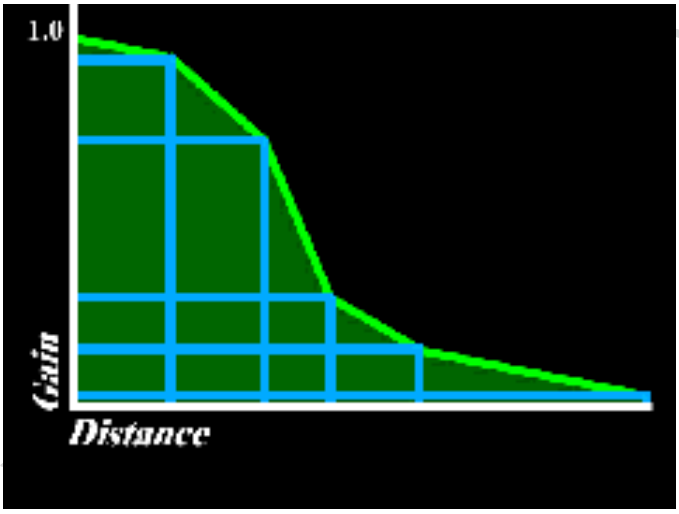

7. `PointSound` extends the `Sound` class

- Similar to `PointLight` in lighting, `PointSound` waves emit radially from a point in all directions.
- Use point sounds to simulate local sounds like:
  - User interface sounds (clicks, alerts)
  - Data sonification
  - Game sounds (laser blasters, monster growls)
- There could be multiple point sounds playing at the same time.

8. Point sound waves are attenuated:

- Amplitude decreases as the viewer moves away
- Attenuation is controlled by a list of value pairs:

- *Distance* from sound position
- *Gain* at that distance



9. PointSound example code

- Load sound data

```
MediaContainer myWave = new MediaContainer("willow1.wav");
```

- Create an attenuation array

```
Point2f[] myAtten = {
   new Point2f( 100.0f, 1.0f ),
   new Point2f( 350.0f, 0.5f ),
   new Point2f( 600.0f, 0.0f )
};
```

- Create a sound

```
PointSound mySound = new PointSound( );
mySound.setSoundData( myWave );
mySound.setEnable( true );
```

```
mySound.setInitialGain( 1.0f );
mySound.setLoop( -1 ); // Loop forever
mySound.setPosition( new Point3f( 0.0f, 1.0f, 0.0f ) )
mySound.setDistanceGain( myAtten );
```

- Set the scheduling bounds

```
BoundingSphere myBounds = new BoundingSphere(
   new Point3d( ), 1000.0 );
mySound.setSchedulingBounds( myBounds );
```

10. `ConeSound` extends the `PointSound` class
    - Similar to `SpotLight` in lighting, `ConeSound` waves emit radially from a point in a direction, constrained to a cone.
    - Use cone sounds to simulate local directed sounds like:

      - Loud speakers
      - Musical instruments
    - There could be multiple cone sounds playing at the same time
11. `ConeSound` extends `PointSound` support for attenuation
    - `ConeSound` uses *two* lists of distance-gain pairs that apply in front and back directions
      - The cones aim direction is the front direction
      - If no back list is given, the front list is used
    - Real-world sound sources emit in a direction
      - Volume (gain) and frequency content varies with angle

- `ConeSound` angular attenuation simulates this effect with a list of angle-gain-filter triples
  - *Angle* from the cones front direction
  - *Gain* at that angle
  - *Cutoff* frequency for a low-pass filter at that angle
- By default, cone sounds are aimed in the positive Z direction with no distance or angular attenuation
  - Attenuation angles are in the range 0.0 to PI radians

12. `ConeSound` example code

- Load sound data

```
MediaContainer myWave = new MediaContainer("willow1.wav");
```

- Create attenuation arrays


```
Point2f[] myFrontAtten = {
   new Point2f( 100.0f, 1.0f ),
   new Point2f( 350.0f, 0.5f ),
   new Point2f( 600.0f, 0.0f )
};
Point2f[] myBackAtten = {
   new Point2f( 50.0f, 1.0f ),
   new Point2f( 100.0f, 0.5f ),
   new Point2f( 200.0f, 0.0f )
};
Point3f[] myAngular = {
   new Point3f( 0.000f, 1.0f, 20000.0f ),
   new Point3f( 0.785f, 0.5f, 5000.0f ),
```

```
    new Point3f( 1.571f, 0.0f, 2000.0f )
};
```

- Create a sound

```
ConeSound mySound = new ConeSound( );
mySound.setSoundData( myWave );
mySound.setEnable( true );
mySound.setInitialGain( 1.0f );
mySound.setLoop( -1 ); // Loop forever
mySound.setPosition( new Point3f( 0.0f, 1.0f, 0.0f ) );
mySound.setDirection( new Vector3f( 0.0f, 0.0f, 1.0f );
mySound.setDistanceGain( myFrontAtten, myBackAtten );
mySound.setAngularAttenuation( myAngular );
```

- Set the scheduling bounds

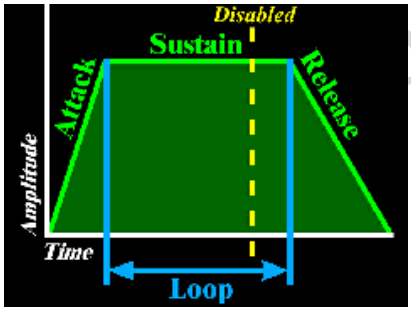

```
BoundingSphere myBounds = new BoundingSphere(
    new Point3d( ), 1000.0 );
mySound.setSchedulingBounds( myBounds );
```

13. Setting scheduling bounds

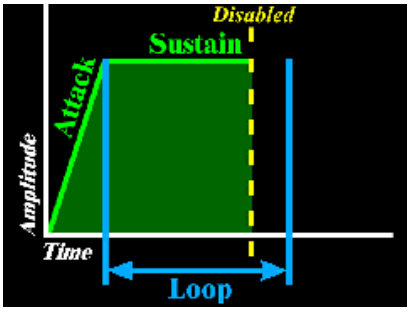
- A sound is hearable (if it is playing) when:
  - The viewers activation radius intersects its scheduling bounds
  - Multiple sounds can be active at once
  - Identical to behavior scheduling
- Sound bounding enables different sounds for different areas of the scene
- By default, sounds have no scheduling bounds and are never hearable!

- **Common error**: forgetting to set scheduling bounds

14. Controlling the sound release when disabling a sound:
    - Enable the release to let the sound finish playing, without further loops
    - Disable the release to stop it immediately

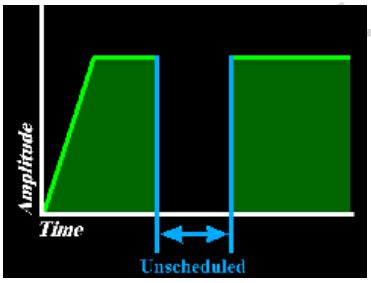


Release enabled

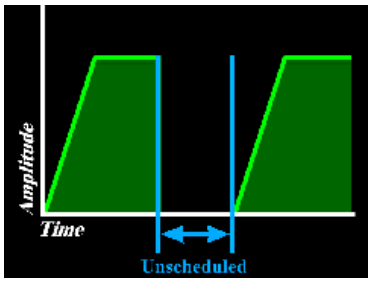


Release disabled

15. Enabling continuous playback when a sound is unscheduled (viewer moves out of scheduling bounds):
    - Enable *continuous* playback to keep it going silently
        - It resumes, in progress if scheduled again
    - Disable *continuous* playback to skip silent playback
        - It starts at the beginning if scheduled again



Continuous enabled



Continuous disabled

16. Sounds can be prioritized
    - Sound hardware and software limits the number of simultaneous sounds
        - Worst case is four point/cone sounds and seven background sounds
    - A low priority sound may be temporarily muted when a high priority sound needs to be played
17. Control features of the environment to enhance realism
    - The `Sound` classes control features of the sound
    - Use `soundscapes` and `aural attributes` to
        - Add reverberation (echos)
        - Use different reverberation for different rooms

        - Control doppler pitch shift
        - Control frequency filtering with distance