Lecture 16

Game Audio
The Role of Audio in Games

Engagement

- **Entertains** the player
  - Music/Soundtrack

- Enhances the **realism**
  - Sound effects

- Establishes **atmosphere**
  - Ambient sounds

- Other reasons?
The Role of Audio in Games

Feedback

• **Indicate** off-screen action
  • Indicate player should move

• **Highlight** on-screen action
  • Call attention to an NPC

• **Increase** reaction time
  • Players react to sound faster

• Other reasons?
History of Sound in Games

Basic Sounds

- Arcade games
- Early handhelds
- Early consoles
Early Sounds: *Wizard of Wor*
History of Sound in Games

- Arcade games
- Early handhelds
- Early consoles
- Starts w/ MIDI
- 5th generation (Playstation)
- Early PCs
History of Sound in Games

- Arcade games
- Early handhelds
- Early consoles

- Starts w/ MIDI
- 5th generation
  (Playstation)
- Early PCs

- Sample selection
- Volume
- Pitch
- Stereo pan
History of Sound in Games

Basic Sounds

Recorded Sound Samples

Some Variability of Samples

More Variability of Samples

- Arcade games
- Early handhelds
- Early consoles

- Starts w/ MIDI
- 5th generation
  (Playstation)
- Early PCs

- Sample selection
- Volume
- Pitch
- Stereo pan

- Multiple samples
- Reverb models
- Sound filters
- Surround sound
The Technical Challenges

- Sound **formats** are not (really) cross-platform
  - It is not as easy as choosing MP3
  - Android, iOS favor different formats

- Sound playback **APIs** are not standardized
  - CUGL is a layer over many different APIs
  - So behavior is not the same on all platforms

- Sound playback crosses **frame boundaries**
  - Mixing sound with animation has challenges
# File Format vs Data Format

## File Format
- The data storage format
  - Has data other than audio
- Many have many encodings
  - `.caf` holds MP3 and PCM
- **Examples:**
  - `.mp3`, `.wav`
  - `.aac`, `.mp4`, `.m4a` (Apple)
  - `.flac`, `.ogg` (Linux)

## Data Format
- The actual audio encoding
  - Basic audio codec
  - Bit rate (# of bits/unit time)
  - Sample rate (digitizes an analog signal)
- **Examples:**
  - MP3, Linear PCM
  - AAC, HE-AAC, ALAC
  - FLAC, Vorbis
# Data Formats and Platforms

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP3</td>
<td>You know what this is</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s alternative lossless codec</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>iLBC</td>
<td>Internet low bit-rate codec (VOIP)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IMA4</td>
<td>Super compression for 16 bit audio</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>µ-law</td>
<td>Like PCM, but optimized for speech</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### The Associated File Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>File Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP3</td>
<td>.mp3</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>Linear PCM</td>
<td>.wav</td>
</tr>
<tr>
<td>MIDI</td>
<td>.mid</td>
</tr>
</tbody>
</table>

- Any other file format is **not cross-platform**

- Apple/iOS is pushing the `.caf` file
  - Stands for Core Audio Format
  - Supports MP3, (HE-)AAC, PCM, ALAC, etc…
  - But not cross-platform
Which Format is Best?

- **MP3** is **not free** to use in games!
  - Patent holders charge a license is on file usage
  - Only free if you sell less than 5000 games

- **Linear PCM** is **completely uncompressed**
  - The file size is huge for music, long sounds
  - Makes it unreasonable for portable use

- **AAC** is (believe it or not) the best
  - Apple charge the license only to the hardware maker
  - But you need iTunes to create the file
Which Format is Best?

- **MP3** is not free to use in games!
  - Patent holders charge a license on file usage
  - Only free if you sell less than 5000 games

- **Linear PCM** is completely uncompressed
  - The file size is huge for music, long sounds
  - Makes it unreasonable for portable use

- **AAC** is (believe it or not) the best
  - Apple charge the license only to the hardware maker
  - But you need iTunes to create the file
Cross-Platform Sound APIs

- **OpenAL**
  - Created in 2000 by Loki Software for Linux
  - Was an attempt to make a sound standard
  - Loki went under; last stable release in 2005
  - Apple supported, but HARD deprecated in iOS 9

- **FMOD**
  - Industry standard for game development
  - Mobile support is possible but not easy
  - Not free; but no cost for low-volume sales
Proprietary Sound APIs

- **Apple AVFoundation**
  - API to support modern sound processing
  - Mainly designed for music/audio creation apps
  - But very useful for games and playback apps

- **OpenSL ES**
  - Directed by Khronos Group (OpenGL)
  - Substantially less advanced than other APIs
  - Really only has support in Android space
What about SDL?

- CUGL is on top of SDL
  - SDL has its own audio API
  - Works on all platforms

- But there are many problems
  - SDL Mixer is ancient (no updates since 2009)
  - No real hardware optimizations at all
  - Significant audio delay due to buffer strategy
  - Features are like OpenAL, not modern APIs
Solution: Cross-Platform Wrappers

- **AudioEngine**: Wrapper to hide the platform
  - **OS X, iOS**: AVFoundation
  - **Android**: OpenSL ES
  - **Windows**: XAudio2 (similar to AVFoundation)
  - **Linux/Other**: SDL Audio

- Limited by the most primitive API
  - In this case SDL Audio (or OpenSL ES)
  - Result is an *last-gen* sound API
Solution: Cross-Platform Wrappers

- **AudioEngine**: Wrapper to hide the platform
  - **OS X, iOS**: AVFoundation
  - **Android**: A/VFoundation (similar to AVFoundation)
  - **Windows**: XAudio2 (similar to AVFoundation)
  - **Linux/Other**: SDL Audio

- Limited by the most primitive API
  - In this case SDL Audio (or OpenSL ES)
  - Result is an *last-gen* sound API

Actually, all but Apple use SDL
Managing Sound Assets

- Sounds can be read from **file** or **preloaded**
  - Preloaded sounds will play immediately
  - Files will *stream* and have a slight playback delay

- Preloading uncompressed the file into **PCM format**
  - Okay (and preferred for) sound effects
  - Inappropriate for music or very large sounds

- This is motivation for our AudioEngine API
  - It separates assets into *sound effects* and *music*
  - Only sound effects can be preloaded
A Word on Streaming

- All sound engines only play PCM data
  - Other files (MP3 etc.) are decoded into PCM data
  - But the data is *paged-in* like memory in an OS

- This is how OGG support was added to CUCL

![Diagram of sound file, sound buffer, and sound engine with arrows indicating append PCM page and retrieve PCM page.](image-url)
Playing a Sound

- Playback may include **multiple sounds**
  - Sounds may play simultaneously (offset)
  - Simultaneous sounds may be same asset
  - **Asset** (source) vs. **Instance** (playback)

- Playback crosses **frame boundaries**
  - It may span multiple animation frames
  - Need to know when it stops playing
  - May need to stop (or pause) it early
Sound Engines are **Mixers**

- Sound Instance
- Sound Instance
- Sound Instance
- **Mixer**

![Sound Engines are Mixers](image)
Classic Model: Channels

Channel
Channel
Channel
...
Channel
Channel
Classic Model: Channels

Engine has fixed number of channels (historically 24)
Classic Model: Channels

Engine has fixed number of channels (historically 24)

Load sound into channel to play it
Classic Model: Channels

- Engine has fixed number of channels (historically 24)
- Load sound into channel to play it

Sound

Mixer

Channel

Channel

Channel

...
Playing a Sound with Channels

- **Request** a sound channel for your asset
  - If none is available, sound fails to play
  - Otherwise, it gives you a id for a channel

- **Load** asset into the channel (but might stream)

- **Play** the sound channel
  - Playing is a property of the channel, not asset
  - Channel has other properties, like volume

- **Release** the channel when the sound is done
  - This is usually done automatically
Application Design

Need to remember channel id
Stopping Sounds

- Would like to know when a sound is finished
  - To free up the channel (if not automatic)
  - To stop any associated animation
  - To start a follow-up sound

- Two main approaches
  - **Polling**: Call an `isPlaying()` method/function
  - **Callback**: Pass a function when play

- **AudioEngine** only allows polling approach
SDL Mixer API

- /**
  * @return channel id for sound playback
  *
  * If no channel is available, returns -1
  * @param channel  The requested channel (-1 for first available)
  * @param chunk    The PCM data (WAV or stream page)
  * @param loops    The number of times to loop the sound
  */
  int Mix_PlayChannel(int channel, Mix_Chunk *chunk, int loops);

- int Mix_HaltChannel(int channel);

- int Mix_FadeOutChannel(int channel, int ms);

- int Mix_Volume(int channel, int volume);

- void Mix_Playing(int channel);

Need to remember channel id
Why This is Undesirable

• Tightly couples architecture to sound engine
  • All controllers need to know this channel id
  • Playback must communicate the id to all controllers

• Instances usually have a semantic meaning
  • Example: Torpedo #3, Ship/crate collision
  • Meaning is independent of the channel assigned
  • Would prefer to represent them by this meaning

• Solution: Refer to instances by keys
The AudioEngine Alternative

- /**
  * Plays given sound as a sound effect (paging out as necessary)
  *
  * @param key the reference key for the sound effect
  * @param sound the sound effect file to play
  * @param loop Whether to loop indefinitely
  * @param volume The sound volume
  */

  void playEffect(string key, const std::shared_ptr<Sound>& sound);

- void stopEffect(string key);

- void setEffectVolume(string key, float volume);

- void getEffectState(string key);
The Latency Problem

- Latency: delay until playback
  - All systems have some
  - Want latency ≤ framerate

- Often a result of buffering
  - Engine has a small buffer
  - Buffer size = playback time
  - Play func writes to buffer
  - Engine reads at size intervals

- Buffering is necessary evil
  - Keeps streams smooth
  - Allows real-time effects
Problem with the Channel Model

• All controls are embedded in the channel
  • **Example**: Volume, looping, play position
  • Restricted to a *predetermined* set of controls

• Modern games want *custom sound-processing*
  • User defined sound filters (low pass, reverb)
  • Advanced equalizer support
  • Support for surround and 3D sound
  • Procedural sound generation
DSP Processing: The Mixer DAG
DSP Processing: The Mixer DAG

Channel model is a special case of this DAG
Example: UDK Kismet

Warehouse Section

Turn on power!
Trigger & Used

Already On?
Compare Sool
In
True
False

WAREHOUSE AREA ON
Play Sound
Play
Cut

WAREHOUSE AREA OFF
Play Sound
Play
Cut

True
Value
Target

False

Delay (0.03)
Start
Finished

Delay (0.03)
Start
Finished

Toggle
Turn On
Turn Off
Target

True

Rattle 1
Rattle 2
Rattle 3
Rattle 4
Rattle 5

Amble 1
Amble 2
Amble 3
Amble 4
Amble 5

Music
Bass

38

Game Audio
AVFoundation Implementation

Load with file or PCM data

AVAudioPlayerNode

AVAudioNode

AVAudioMixerNode

AVAudioPlayerNode

AVAudioNode
Provided AVAudioNodes

- **AVAudioUnitEQ**
  - Support for equalizer and low pass/high pass filters

- **AVAudioUnitDistortion**
  - Support for custom distortion effects

- **AVAudioUnitReverb**
  - Support for custom reverb effects

- **AVAudioEnvironmentNode**
  - Support for *positional* and 3D audio
Positional Audio: Surround Sound

Sub

Left Front

Center

Right Front

Left Surround

Left Rear Surround

Center

Right Rear Surround

Player

Game Audio
Positional Audio: Surround Sound

Sub

Left Front

Center

Right Front

Left Surround

Left Rear Surround

Right Rear Surround

Game Audio
Positional Audio: Surround Sound

Original source must be mono to work properly

Game Audio
AVAudioEnvironmentNode API

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Location of sound source</td>
</tr>
<tr>
<td>listenerPosition</td>
<td>Location of the listener</td>
</tr>
<tr>
<td>listenerVectorOrientation</td>
<td>Facing orientation of the listener</td>
</tr>
<tr>
<td>obstruction</td>
<td># decibels to reduce sound from source (Affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>occlusion</td>
<td># decibels to reduce sound from source (Affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>reverbBlend</td>
<td>Amount reverb to add to scene</td>
</tr>
<tr>
<td>renderingAlgorithm</td>
<td>The type of rendering algorithm to use</td>
</tr>
</tbody>
</table>
# AVAudioEnvironmentNode API

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Location of sound source</td>
</tr>
<tr>
<td>listenerPosition</td>
<td>Location of the listener</td>
</tr>
<tr>
<td>listenerVectorOrientation</td>
<td>Facing orientation of the listener</td>
</tr>
<tr>
<td>obstruction</td>
<td># decibels to reduce sound from source (affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>occlusion</td>
<td># decibels to reduce sound from source (affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>reverbBlend</td>
<td>Amount reverb to add to scene</td>
</tr>
<tr>
<td>renderingAlgorithm</td>
<td>The type of rendering algorithm to use</td>
</tr>
</tbody>
</table>

- **Obstruction** and **Occlusion** do not compute physics.
Modeling Sound Environments

Must compute the obstacle/occlusion values separately
Advanced: Reverb Calculations

- Uses audio raytracing
- Also material reflection
- No AVFoundation support
**Advanced: Reverb Calculations**

- Uses audio raytracing
- Also material reflection
- No AVFoundation support

Area of active development in AAA games
### AVAudioEnvironmentNode API

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Location of sound source</td>
</tr>
<tr>
<td>listenerPosition</td>
<td>Location of the listener</td>
</tr>
<tr>
<td>listenerVectorOrientation</td>
<td>Facing orientation of the listener</td>
</tr>
<tr>
<td>obstruction</td>
<td># decibels to reduce sound from source (Affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>occlusion</td>
<td># decibels to reduce sound from source (Affects direct sound, but not reverb)</td>
</tr>
<tr>
<td>reverbBlend</td>
<td>Amount reverb to add to scene</td>
</tr>
<tr>
<td>renderingAlgorithm</td>
<td>The type of rendering algorithm to use</td>
</tr>
</tbody>
</table>

So what does this actually do?
AVAudio3DMixingRenderingAlgorithm

- AVAudio3DMixingRenderingAlgorithmStereoPassThrough
  - **Turns off positional** rendering and uses source encoding

- AVAudio3DMixingRenderingAlgorithmEqualPowerPanning
  - Pans the volume across **two stereo channels**

- AVAudio3DMixingRenderingAlgorithmSoundField
  - Positional audio for **surround sound**

- AVAudio3DMixingRenderingAlgorithmSphericalHead
  - **Binaural synthesis** assuming a spherical head

- AVAudio3DMixingRenderingAlgorithmHRTF
  - **Binaural synthesis** with the Head Related Transfer Function
Binarual Synthesis

- Positional sound is fakey
  - Essentially volume control
  - Cannot pinpoint source
- **Goal**: realistic perception
  - Track the sound parallax
  - Account for shape of head
- Limited to headphones
  - Cannot do speakers (yet)
- **Example**: Papa Sangre

Image coordinates $(r, \theta, \phi)$

(Left, Right) $x_L(t)$, $x_R(t)$

(dry source) $x_A(t)$

$\tilde{h}_L(t)$, $\tilde{h}_R(t)$

HRIR database, interpolation

(perceived image location)
Example: Papa Sangre

SEE WITH YOUR EARS. 
MOVE WITH YOUR FEET.
Summary

• Audio design is about creating soundscapes
  • Music, sound effects, and dialogue
  • Combining sounds requires a sound engine

• Cross-platform support is a problem
  • Licensing issues prevent a cross-platform format
  • Very little standardization in sound APIs

• Best engines use digital signal processing (DSP)
  • Mixer graph is a DAG supporting sound effects
  • Some limited support for positional audio