Lecture 18

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?**

Game Audio
History of Sound in Games

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

- Basic Sounds
- Recorded Sound Samples
- Sample = pre-recorded audio

- 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
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
  - Different platforms favor different formats

- Sound playback **APIs** are not standardized
  - LibGDX & CUGL are layered over many APIs
  - 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`, `.aiff`
- `.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
### Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 largely avoided due to patent issues.
# Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 largely avoided due to patent issues.
# Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td>NOT SOUND; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 largely avoided due to patent issues.

Supported in CUGL
Which Formats Should You Choose?

- **Question 1:** Streaming or no streaming?
  - Audio gets large fast; music often streamed
  - But streaming creates overhead; bad for sound fx
  - Few engines support WAV streams (LibGDX & CUGL do)

- **Question 2:** Lossy or lossless compression?
  - Music can be lossy; sound fx not so much
  - Only FLAC and WAV are standard lossless

- **Question 3:** How many channels (speakers) needed?
  - MP3 channel is *stereo only*
  - Others support many channels (e.g. 7.1 surround)
Which Formats Should You Choose?

- **Question 1**: Streaming or no streaming?
  - Audio gets large fast; music often streamed
  - But streaming creates overhead; bad for sound fx
  - Few engines support WAV streaming (CUGL does)

- **Question 2**: Lossy or lossless compression?
  - Music can be lossy; sound fx not so much
  - Only FLAC and WAV are standard lossless

- **Question 3**: How many channels (speakers) needed?
  - MP3 channel is *stereo only*
  - Others support many channels (e.g. 7.1 surround)

**Sound FX**: Linear PCM/WAV

**Music**: OGG Vorbis
Linear PCM Format

- Sound data is an array of **sample** values

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>-0.1</td>
<td>0.3</td>
<td>-0.5</td>
<td>0.0</td>
<td>-0.2</td>
<td>-0.2</td>
<td>0.0</td>
<td>-0.6</td>
<td>0.2</td>
<td>-0.3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

- A sample is an **amplitude** of a sound wave

- Values are normalized -1.0 to 1.0 (so they are floats)
Linear PCM Format

- Sound data is an array of **sample** values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- A sample is an **amplitude** of a sound wave

- Sometimes encoded as shorts or bytes MIN to MAX
Linear PCM Format

- Sound data is an array of **sample** values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume
Linear PCM Format

- Sound data is an array of **sample** values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume
Linear PCM Format

- Samples are organized into (interleaved) **channels**

  | 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | 0.0 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- Each channel is essentially a **speaker**
  - Mono sound has one channel
  - Stereo sound has two channels
  - 7.1 surround sound is **eight** channels

- A **frame** is set of simultaneous samples
  - Each sample is in a separate frame
Linear PCM Format

- The sample rate is frames per second

- Example: 0.5 seconds of stereo at 44.1 kHz
  - $0.5 \text{ s} \times 44100 \text{ f/s} = 22050 \text{ frames}$
  - $2 \text{ samples/frame} \times 22050 \text{ frames} = 44100 \text{ samples}$
  - $4 \text{ bytes/sample} \times 44100 \text{ samples} = 176.4 \text{ kBytes}$

- 1 minute of stereo CD sound is 21 MB!
Playing Sound Directly

Game Loop

PCM data buffer

Sound Card
Playing Sound Directly

Game Loop

Write PCM chunk to buffer

PCM data buffer

Sound Card

Game Audio
Direct Sound in CUQL: AudioNode

- Class representing an audio **source instance**
  - Not the same as Sound, which is an asset
  - sound->createNode() returns an instance node
  - Plug node into an AudioOutput (device)

- Data is read from method

```cpp
/**
 * Reads up to the specified number of frames into the given buffer
 *
 * @param buffer The read buffer to store the results
 * @param frames The maximum number of frames to read
 */
Uint32 AudioNode::read(float* buffer, Uint32 frames);
```
Direct Sound in CUGL: AudioNode

- Class representing an audio **source instance**
  - Not the same as `Sound`, which is an **asset**
  - `sound->createNode()` returns an instance node
  - Plug node into an `AudioOutput`

- Data is read from method
  ```cpp
  /**
   * Reads up to the specified number of frames into the given buffer
   *
   * @param buffer The read buffer to store the results
   * @param frames The maximum number of frames to read
   */
  Uint32 AudioNode::read(float* buffer, Uint32 frames);
  ```

Called in separate **audio thread**
The Latency Problem

- Buffer is really a queue
  - Output from queue front
  - Playback writes to end
  - Creates a playback delay

- Latency: amount of delay
  - Some latency must exist
  - Okay if latency $\leq$ framerate
  - Android latency is $\sim 90$ ms!

- Buffering is a necessary evil
  - Keeps playback smooth
  - Allows real-time effects
Playing Sound Directly

Choice of buffer size is important!

- **Too large**: long latency until next sound plays
- **Too small**: buffers swap too fast, causing audible pops
Playing Sound Directly

- **Game Loop**
- **Sound Card**

**Write PCM chunk to buffer**

**PCM data buffer**

- **Windows**: 528 bytes (even if you ask for larger)
- **MacOS, iOS**: 512-1024 bytes (hardware varies)
- **Android**: 2048-4096 bytes (hardware varies)
How Streaming Works

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

- Why LibGDX/CUGL can stream WAV files too!
How Streaming Works

- **Sound**: Sound asset that is *preloaded* as full PCM
- **Music**: Sound asset that is *streamed* as PCM pages
Handling Multiple Sounds

Literally!

Sound Card

PCM Data

PCM Data

PCM Data

PCM Data
Handling Multiple Sounds

- Can create values outside of -1 to 1
  - This causes clipping/distortion
  - Common if many simultaneous sounds
- Audio engineer must balance properly

Literally!

PCM Data

Sound Card
Why is Mixing Hard?

- 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
We Want Something Simpler!

- Want ability to **play** and **track** sounds
  - Functions to load sound into card buffer
  - Functions to detect if sound has finished

- Want ability to **modify** active sounds
  - Functions for volume and pitch adjustment
  - Functions for stereo panning (e.g. left/right channels)
  - Functions to pause, resume, or loop sound

- Want ability to **mix** sounds together
  - Functions to add together sound data quickly
  - Background process for dynamic volume adjustment
We Want Something Simpler!

- Want ability to **play** and **track** sounds
  - Functions to load sound into card buffer
  - Functions to detect if sound has finished

- Want ability to **modify** active sound
  - Functions for volume and pitch adjustment
  - Functions for stereo panning (e.g. left/right channels)
  - Functions to pause, resume, or loop sound

- Want ability to **mix** sounds together
  - Functions to add together sound data quickly
  - Background process for dynamic volume adjustment

This is the purpose of a **sound engine**
Cross-Platform Sound Engines

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

• 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
Proprietary Sound Engines

- Apple AVFoundation
  - API to support modern sound processing
  - Mainly designed for music/audio creation apps
  - By Apple
  - And many competing 3rd party solutions

- 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 it is a **extremely** low-level API
  - Fill the buffer with linear PCM data
  - Either pull (callback) or push (queue)
  - No support for non-WAV audio formats
  - No support for mixing, pausing, or anything
Solution: CUCL Audio Classes

- **AudioChannels**: Simple audio interface
  - Essentially uses the OpenAL model
  - Very easy to use and understand
  - Limited to pre-recorded sound files

- **AudioManager**: Advanced audio interface
  - Direct access to the *audio filter graph*
  - Requires a lot of audio knowledge to use
  - Can support complex audio assets (patches)
Classic Model: **Channels**

- Channel
- Channel
- Channel
- Channel
- ...
- Channel

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**

- **Queue to follow after**
- **Sound**
- **Sound**
- **Mixer**
- **Engine has fixed number of channels (historically 24)**
- **Load sound into channel to play it**
Playing a Sound with Channels

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

Volume is property of channel!
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*
Application Design

How AudioChannels works!

Assign this a key identifier
The AudioChannels API

- /**
  * 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);
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 listener to the engine

• **AudioChannels** allows both approaches
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
Example: UDK Kismet
Example: FMOD
Example: Pure Data

Timbre Implementation

Game Audio
Channel Model is a Special Case

Interface to set state: volume, pan, fadeout
Channel Model is a Special Case

Input has scheduling features as well

All happens behind scenes of AudioChannels interface.
Channel Model is a Special Case

Theoretically input should accept any **audio subgraph**
An Observation About “Channels”

Scheduler accepts any **audio subgraph**
An Observation About “Channels”

Scheduler accepts any audio subgraph

But AudioChannels does not expose this, yet!
Current Interface for AudioChannels

- /**
  * Plays given sound effect, and associates it with the key.
  *
  * @return true if there was an available channel for the sound
  */
  
  bool playEffect(std::string key, const std::shared_ptr<Sound>& sound)

- /**
  * Plays given music asset as a background track.
  *
  * Music is handled differently from sound effects.
  * You can only play one music asset at a time.
  */
  
  void playMusic(const std::shared_ptr<Sound>& music)
Plan for Next Year

- /**
  * Plays given audio node, and associates it with the key.
  *
  * @return true if there was an available channel for the node
  */
  
  bool play(string key, const shared_ptr<AudioNode>& node)

- /**
  * Plays given audio node in the pre-specified channel
  *
  * Channels are separated into fixed and floating (for Sound FX).
  * Channel should be a fixed channel.
  */
  
  void play(Uint 32 channel, const shared_ptr<AudioNode>& node)
Creating Your Own Audio Graph

• **Class **AudioManager
  • Starts/stops audio system
  • Specifies the buffer size
  • Provides factory methods
  • Allocates input and output

• **Class **AudioOutput
  • Terminal node of the graph
  • Can be named or default
  • Defines the # of channels
  • Defines the sample rate
Creating Your Own Audio Graph

AudioNode → AudioMixer → AudioFader → AudioOutput

Needed for click-free stopping and pausing
AudioNode Classes in CUGL

- **AudioPlayer**
  - Single playable instance for a sound asset

- **AudioFader**
  - Fade-in, fade-out and cross-fade effects

- **AudioPanner**
  - Simple stereo channel panning

- **AudioInput**
  - A recording node, for real-time playback
Advanced: Surround Sound

Sub

Left Front

Center

Right Front

Left Surround

Player

Right Surround

Left Rear Surround

Game Audio

Right Rear Surround
Advanced: Surround Sound

Sub
Left Front
Center
Right Front

Left Rear Surround
Left Surround
Center
Right Rear Surround

Game Audio
Advanced: Surround Sound

Original source must be mono to work properly
Advanced: Surround Sound

Original source must be mono to work properly

See AudioSpinner
Advanced: Reverb Calculations

- Uses audio raytracing
- Also material reflection
- Potential MEng project
Advanced: Reverb Calculations

- Uses audio raytracing
- Also material reflection
- Potential MEng project

Area of active development in AAA games
Advanced: Binarual Synthesis

- Mobile positional sound?
  - Only stereo: left/right
  - Cannot pinpoint source

- **Goal**: realistic perception
  - Track the sound parallax
  - Account for shape of head

- Not (yet) in CUGL
  - In experimental branch
  - Will merge in summer

![Diagram of sound localization](image-url)
Example: *Papa Sangre*
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
  - CUML has some early support for all this