Lecture 12

Game Audio
The Role of Audio in Games

Engagement

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

- Enhances the **realism**
  - Sound effects

- Establishes **atmosphere**
  - Ambient sounds
The Role of Audio in Games

Feedback

- **Indicate** off-screen action
  - Hint for player action
- **Highlight** on-screen action
  - Call attention to an NPC
- **Increase** reaction time
  - Players react to sound faster
History of Sound in Games

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

Sample = pre-recorded audio

- Arcade games
- Early handhelds
- Early consoles
- Starts w/ MIDI
- 5th generation
  (Playstation)
- Early PCs

Basic Sounds → Recorded Sound Samples
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 (Playstation)
- 5th generation
- 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

LibGDX is here

CUGL is here

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

<table>
<thead>
<tr>
<th><strong>File Format</strong></th>
<th><strong>Data Format</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>● The data storage format</td>
<td>● The actual audio encoding</td>
</tr>
<tr>
<td>● Has data other than audio</td>
<td>● Basic audio codec</td>
</tr>
<tr>
<td>● Many have many encodings</td>
<td>● Bit rate (# of bits/unit time)</td>
</tr>
<tr>
<td>● .caf holds MP3 and PCM</td>
<td>● Sample rate (digitizes an analog signal)</td>
</tr>
<tr>
<td><strong>Examples:</strong></td>
<td><strong>Examples:</strong></td>
</tr>
<tr>
<td>● .mp3, .wav, .aiff</td>
<td>● MP3, Linear PCM</td>
</tr>
<tr>
<td>● .aac, .mp4, .m4a (Apple)</td>
<td>● AAC, HE-AAC, ALAC</td>
</tr>
<tr>
<td>● .flac, .ogg (Linux)</td>
<td>● FLAC, Vorbis</td>
</tr>
</tbody>
</table>
## 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.
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

| 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

- 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 | 0.0 | -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 | -0.2 | 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 -0.2 0.0 -0.6 0.2 -0.3 0.4 0.0

  **frame**

- 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 s * 44100 f/s = 22050 frames
- 2 samples/frame * 22050 frames = 44100 samples
- 4 bytes/sample * 44100 samples = 176.4 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
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 (device)

• Data is read from method

/**
 * 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 ≤ framerate
  - Android latency is ~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

- Write PCM chunk to buffer
- PCM data buffer

---

Game Loop

- 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
How Streaming Works

- **Sound File**: Sound asset that is preloaded as full PCM...
- **Streaming Buffer**: 
  - **Page size** set by file format
  - **Chunk size** set by audio API
- **Sound Card**: Sound asset that is streamed as PCM pages

- **Sound**: 
  - LibGDX distinction; less true in CUGL
- **Music**: 
  - Sound asset that is streamed as PCM pages
Handling Multiple Sounds

![Diagram showing the process of handling multiple sounds](image)

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!
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 to adjust volume and pitch
  - 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
  - Google is deprecating in 2022
Proprietary Sound Engines

- **Apple AVFoundation**
  - API to support modern sound processing
  - Mainly designed for music/audio creation apps
  - By Apple

- **OpenSL ES**
  - Directed by Khronos Group (OpenGL)
  - Substantially less advanced than other APIs
  - Really only has support in Android space
  - Google is deprecating in 2022

And many competing 3rd party solutions
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: CUGL Audio Classes

- **AudioEngine**: Playing sound effects
  - Built on the OpenAL model
  - Very easy to use and understand
  - Designed for simultaneous sounds

- **AudioQueue**: Playing music sequences
  - Accessed from the AudioEngine
  - Creates seamless playback queues
  - Ideal for long-running music loops
Solution: CUGL Audio Classes

- **AudioEngine**: Playing sound effects
  - Built on the OpenAL model
  - Very easy to use and understand
  - Modern version of OpenAL model

- **AudioQueue**: Playing music sequences
  - Accessed from the AudioEngine
  - Creates seamless playback queues
  - Ideal for long-running music loops
Classic Model: Playback Slots

Engine has fixed number of slots (historically 24)
Classic Model: Playback Slots

Engine has fixed number of slots (historically 24)

Load sound into a slot to play it

Slot
Slot
Slot
...
Slot
Sound

Mixer
Classic Model: Playback Slots

Engine has fixed number of slots (historically 24)

Queue to follow after

Load sound into a slot to play it
Playing a Sound with Slots

• **Request** a playback slot for your asset
  • If none is available, sound fails to play
  • Otherwise, it gives you an id for the slot

• **Load** asset into the slot (but might stream)

• **Play** the playback slot
  • Playing is a property of the slot, not asset
  • Playback slot has other properties, like volume

• **Release** the slot when the sound is done
  • This is usually done automatically
Application Design

Need to remember the slot id

Volume is property of a slot!
Why This is Undesirable

- Tightly couples architecture to sound engine
  - All controllers need to know this playback slot id
  - Playback must communicate id to all controllers
- Instances usually have a *semantic meaning*
  - **Example**: Torpedo #3, Ship/crate collision
  - Meaning is independent of the slot assigned
  - Would prefer to represent them by this meaning
- **Solution**: Refer to instances by *keys*
Application Design

How AudioEngine works!

Slot
Slot
Slot
...
Slot

Assign this a **key** identifier
The AudioEngine API

- /**
   * Plays the given sound, and associates it with the specified key.
   *
   * @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 play(const string key, const std::shared_ptr<Sound>& sound);

- void stop(const string key);

- void setVolume(const string key, float volume);

- void getState(const string key);

Refer to instance logically
Stopping Sounds

- Would like to know when a sound is finished
  - To free up the slot (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

- `AudioEngine` allows both approaches
Gapless Playback

- Gapless playback requires a **queue**
  - Queue immediately plays next sound on completion
  - Ideally with some **crossfade** to prevent pops

- Supported by class **AudioQueue**
  - Built on top of AudioEngine; use `allocQueue()` method
  - Permanently takes over a slot for the queue
  - Can have multiple queues – as many as there are slots
  - But no simultaneity guarantee between queues

- **AudioQueue** is *kind of* similar to AudioEngine
  - But no need for keys, as there is only one slot
### The AudioQueue API

- void enqueue(const std::shared_ptr<Sound>& sound);

- void advance(unsigned int steps);

- void setVolume(float volume);

- void getState();
Problem with the Slots Model

- All controls are embedded in the slot
  - 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

- Source → Effect → Effect → Effect
- Source → Effect → Mixer → Effect
- Source → Mixer → Effect → Main Mixer
Example: UDK Kismet

Warehouse Section

WAREHOUSE AREA ON
- Play Sound
  - Play
  - Cut
- In
- Value Target
- True

WAREHOUSE AREA OFF
- Play Sound
  - Play
  - Cut
- In
- Value Target
- False

Delay (0.03)
- Start
- Finished

Toggle
- Turn On
- Turn Off
- Target

rattle 1
- music
- bass
- Ambience 1

rattle 2
- music
- bass
- Ambience 2

rattle 3
- music
- bass
- Ambience 3

rattle 4
- music
- bass
- Ambience 4

rattle 5
- music
- bass
- Ambience 5

Turn on power
- Trigger & Used
- Used

Already On?
- Compare Bool
  - True
  - False

Game Audio
Example: FMOD
Example: Pure Data
The Slot Model is a Special Case

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

Input has scheduling features as well

Main Mixer

All happens behind scenes of AudioEngine interface.
The Slot Model is a Special Case

Theoretically input should accept any **audio subgraph**
The AudioEngine Revisited

- /**
  * Plays the given sound, and associates it with the specified key.
  *
  * @param key       the reference key for the sound effect
  * @param node     the audio node to play
  * @param loop      whether to loop indefinitely
  * @param volume  the sound volume
  */
  void play(const string key, const std::shared_ptr<AudioNode>& node);

- void stop(const string key);
- void setVolume(const string key, float volume);
- void getState(const string key);
The AudioEngine Revisited

- /**
  * Plays the given sound, and associates it with the specified key.
  *
  * @param key       the reference key for the sound effect
  * @param node     the audio node to play
  * @param loop      whether to loop indefinitely
  * @param volume  the sound volume
  */
  void play(const string key, const std::shared_ptr<AudioNode>& node);

- void stop(const string key);

- void setVolume(const string key, float volume);

- void getState(const string key);

Also supported by AudioQueue

Refer to instance logically
Using AudioNode in AudioEngine

- Normal playback is **built on top of it**
  - Uses `sound->getInstance()` to get your node
  - So just as fully featured as normal playback

- But the node must implement `completed()`
  - This is optional method for `AudioNode` subclasses
  - The default implementation always returns `false`
  - But that means the sound never finished playing
  - So the scheduler cannot free slot for new sound
AudioNode Classes in CUCL

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

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

- **AudioMixer**
  - Group several *simultaneous* nodes together

- **AudioScheduler**
  - Used to queue up sounds in a *sequence*
AudioNode Classes in CUCL

- **AudioPanner**
  - Simple stereo channel panning

- **AudioSpinner**
  - Like panner but works on 7.1 sound fields

- **AudioResampler**
  - Converts audio to different sample rate

- **AudioSynchronizer**
  - Experimental beat detection for rhythm games
Application: **Vertical Layering**

- Create with `getInstance()`
- Assign to slot
- Source
- Source
- Source
- Source

**Mixer** → **Slot**
Application: Vertical Layering

Control volume individually

Create with getInstance()

Assign to slot

Source Source Source Source

Mixer

Slot
Application: Vertical Layering

AudioMixer completes when all of its input nodes do

Create with getInstance()
Two Special AudioNodes

- **Class AudioOutput**
  - Terminal node of the graph
  - Represents output device
  - Can be *named* or *default*
  - Defines channels, sample rate

- **Class AudioInput**
  - Initial node of the graph
  - Represents input device
  - Can be *named* or *default*
  - May or may not match output
Two Special AudioNodes

- **Class AudioOutput**
  - Terminal node of the graph
  - Represents output device
  - Can be named or default
  - Defines channels, sample rate

- **Class AudioInput**
  - Initial node of the graph
  - Represents input device
  - Can be named or default
  - May or may not match output
Two Special AudioNodes

- **Class AudioOutput**
  - Terminal node of the graph
  - Represents output device
  - Can be *named* or *default*
  - Defines channels, sample rate

- **Class AudioInput**
  - Initial node of the graph
  - Represents input device
  - Can be *named* or *default*
  - May or may not match output

These are all managed by the AudioDevices singleton
Advanced: Reverb Calculations

- Uses audio raytracing
- Also material reflection
- In many AAA games
Advanced: Reverb Calculations

- Uses audio raytracing
- Also material reflection
- In many AAA games

Unfortunately this is a patent mine field!
Advanced: Surround Sound

Sub
---

Left Front

Center

Right Front

Left Surround

Player

Left Rear Surround

Game Audio

Right Rear Surround
Advanced: Surround Sound
Advanced: Surround Sound

Original source must be mono to work properly

Sub
Left Front
Center
Right Front

Left Rear Surround
Left Surround

Right Rear Surround
Game Audio
Advanced: Binaural 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
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
  • CUGL has some early support for all this