Optimizing Audio for Mobile Development

Ben Houge
Berklee College of Music
Music Technology Innovation
Valencia, Spain

GAME DEVELOPERS CONFERENCE™ CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 19–21, 2014

Optimizing Audio

- Different from movies or CD’s
- Data size vs. CPU usage
- For every generation, we must optimize
- Maximize quality within constraints

Leisure Suit Larry: Love for Sail!

- MS-DOS 5.2 or Windows 3.1 or greater
- 486 DX2 processor (66 MHz)
- 8 MB RAM (DOS), 12MB (Windows)
- 22 MB Hard drive space
- SVGA, 256 colors
- 1996 specs!

Compare with iPhone 6

- 1 GB of RAM
- 64 bit, dual core 1.38 GHz processor
- 16-128 GB of hard drive space
- 2014 specs!

Optimizing Audio

- Efficient Spotting and Editing for Games
- Digital Audio Theory and Formats
- Creative Reuse of Audio
- File Management and Middleware

Efficient Spotting and Editing for Games

The best way to save space on audio in your game:

Don’t put audio in your game!

Spotting a Game

- Look at the information the game knows
- Think about what’s important to the user
- Don’t assume you need audio all the time
- Prototype and iterate
- Log your sounds and analyze
- Get an audio professional involved early

Editing Sound for a Game

- Trim sounds tightly
- Edit to zero crossings
- Record loud, turn down in software

Audio Quality and File Size

Three Parameters Affect Bitrate

- Sample rate
- Bit depth
- Number of channels

Sample Rate

- Sampling an analog signal from a mic
- Measurement of amplitude
- Linear PCM (uncompressed)
- Sampling rate: how often we sample
- The Nyquist frequency
- CD quality is 44100 Hz

Bit Depth

- Resolution of each sample measurement
- Lower bit depths increase noise
- CD quality is 16 bits

Sample Rate/Bit Depth Demo

Perceptual Audio Coding

- Analyze a signal, throw away frequencies we won’t miss.
- Lossy compression: some data is gone forever
- Varies based on input: noise compresses very little, silence compresses a lot

Examples

- MP3
- Ogg Vorbis
- AAC
- WMA/XMA
- AC3

Perceptual Audio Coding

- Sometimes this can be used for aesthetic purposes
- Performance hit for decompressing sounds
- Hardware acceleration available on some platforms (iPhone, Xbox One...)

Lossless Audio Compression

- Examples
  - FLAC
  - Apple Lossless
- All data is retained
- Not ideally suited for games

Compress with Care

- Don’t use the same settings for all sounds
- Takes more time, but improves quality
- Keep hi-res version, work from copy

Creative Reuse of Sounds

Benefits of Reuse

- Create new sounds from existing data
- Minimize footprint, maximize variation
- Map audio parameters on to game data
- Close synchronization and immersion

Example: footstep sounds

- Choosing from a set of sounds randomly
- Real-time manipulation of pitch, volume
- Emulating the physics of the natural world
- Very efficient in terms of CPU

Example: Combinatoriality

- Multiple layers of random sounds
- Exponential increase in combinations
- Consistency also increases
- Still very efficient

Example: Randomizing Loops

- Infinite variation from a short loop
- Scalability and parameterization
- A kind of granular synthesis

Example: Generative Music

- Multiple independent layers
- Intermittent phrases
- Synchronized to a metronome
- Cued from game events
- Scalable to gameplay

Example: EndWar Loading Music

- Extremely limited resources
- Little memory, no disk access
- Music responded to load times
- xxx kb

Real-Time Effects

- Sophisticated digital signal processing
- Avoids the need for multiple versions
- Scalable with game parameters
- Infinite variety

Effects in EndWar

- Filter and distortion for radio effect
- Filter for distance simulation
- Audio particle system for explosions

Audio Effects in iOS7

kAudioUnitSubType_BandPassFilter
kAudioUnitSubType_DynamicsProcessor
kAudioUnitSubType_Delay
kAudioUnitSubType_AUFilter
kAudioUnitSubType_GraphicEQ
kAudioUnitSubType_HighPassFilter
kAudioUnitSubType_HighShelfFilter
kAudioUnitSubType_PeakLimiter
kAudioUnitSubType_LowPassFilter
kAudioUnitSubType_LowShelfFilter
kAudioUnitSubType_MultiBandCompressor
kAudioUnitSubType_MatrixReverb
kAudioUnitSubType_NetSend
kAudioUnitSubType_ParametricEQ
kAudioUnitSubType_SampleDelay
kAudioUnitSubType_Pitch

Even easier to use with new AVAudioEngine in iOS8!

Synthesis

- Huge topic with much potential
- Like “soft synths” in music production
- More CPU usage, but far less data
- Opportunities for real-time manipulation
- A golden age of MIDI?
- Also for sound effects

File Management and Middleware

Data management for game audio

- Requires organization
- Tracking many small files
- Enforce a naming convention

Audio Engine vs. Game Engine

- Game engine calls sound events
- Audio engine manages sound events
- Audio designer defines audio behavior
- Clear and efficient division of labor

Possible Audio Behaviors

- Play sound, Play multiple sounds, Play sound with variations, Stop sound, Play one sound while stopping another, Change volume on sound, Apply effect, etc.
- Audio implementer doesn’t need to know about this; just call the event

Use Middleware

- Most common solutions
  - Wwise (Audio Kinetic)
  - FMOD
- Big gains for all but the simplest games
- Reasonably priced (in some cases free)
- Available for Android, iOS, and others

Middleware Advantages

- Mix in-game for a tight iteration loop
- Log and profile your audio data use
- Avoid redundancies
- Export for multiple platforms
- Prioritize your sounds
- Incorporate effects plug-ins

Conclusion

Summary

- Spot your game carefully for audio
- Edit your sounds tightly
- Compress sounds individually
- Reuse your sounds creatively
- Track your assets carefully
- Mix and profile your sounds iteratively

Questions?

Ben Houge

Berklee College of Music

Music Technology Innovation

bhouge@berklee.edu

Twitter/微博: @AleaBoy

- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level