Fundamentals of Audio Programming

Bjorn Roche
XO Audio, LLC

Who Am I?

- Software Designer
- Consultant
  - Sterling Sound
  - Z-Systems
  - Indaba

Who Am I?

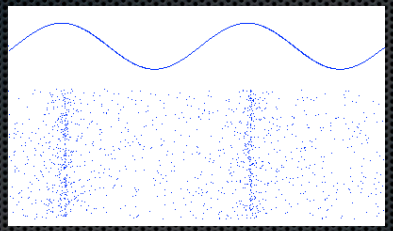
I developed a web-based audio editor called “Mantis” for Indaba Music.
http://www.indabamusic.com/landing/mantis

Who Am I?

Developing a new Audio Editor that lets you collaborate in real-time from anywhere on the globe.
http://www.xonami.com

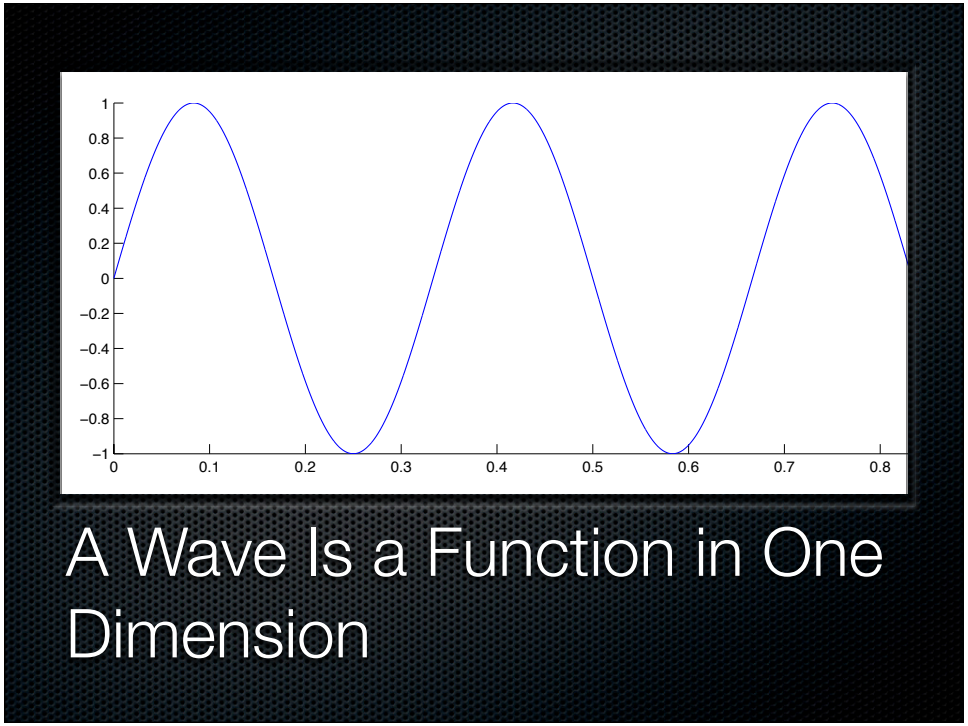
- What is Sound? What is sound on a computer? (Waves, Sampling)
- How do we get sound in and out of a computer? (Callback and Blocking I/O)
- How do we keep sound playback smooth and uninterrupted? (Buffering)
- How does audio playback work? (Inter-thread communication)
- How do we synchronize audio and video in software? On the web? (HTML 5/Javascript)
- How do we synchronize audio and other media? (Master Clocks)
- How do we manipulate sound? (DSP)

What is Sound?

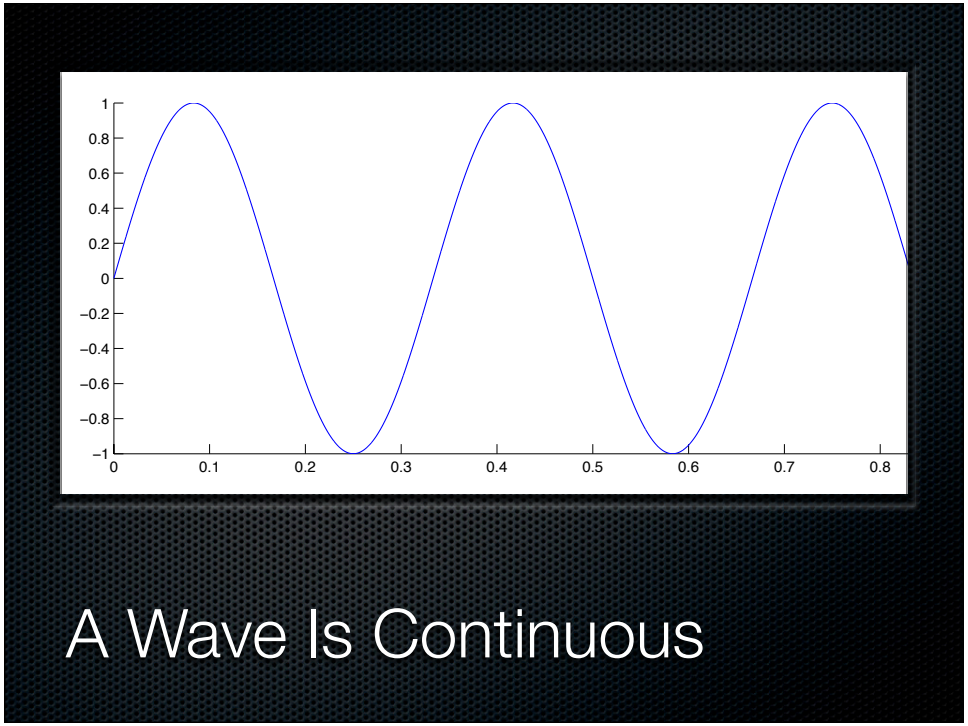


- We don't really need to know that.
- For us, it's a wave.

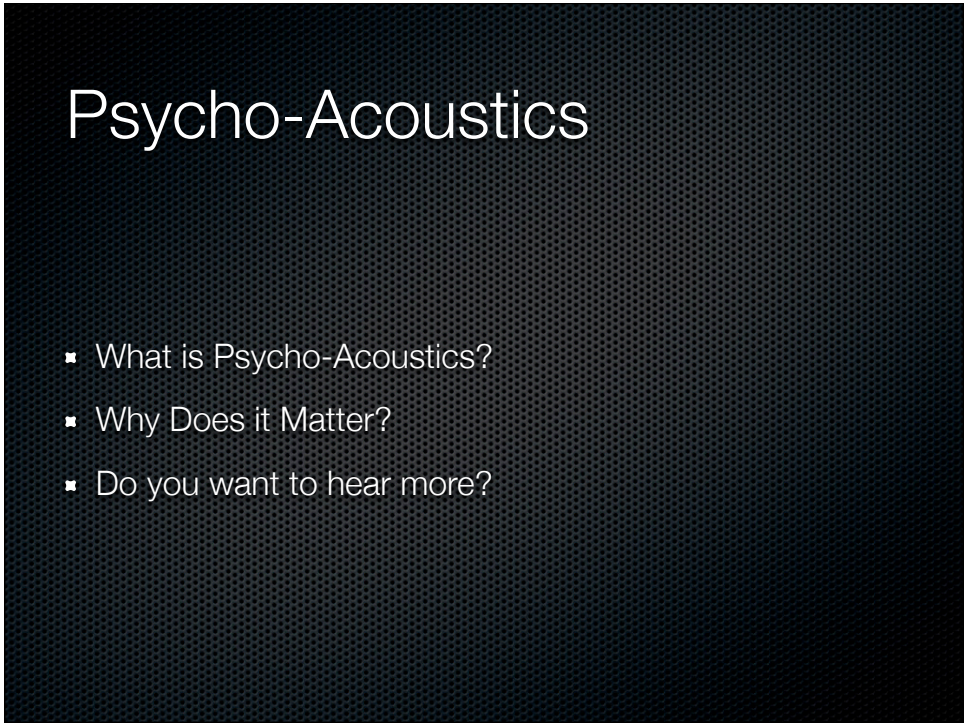
There's a lot we just don't need to know about sound.



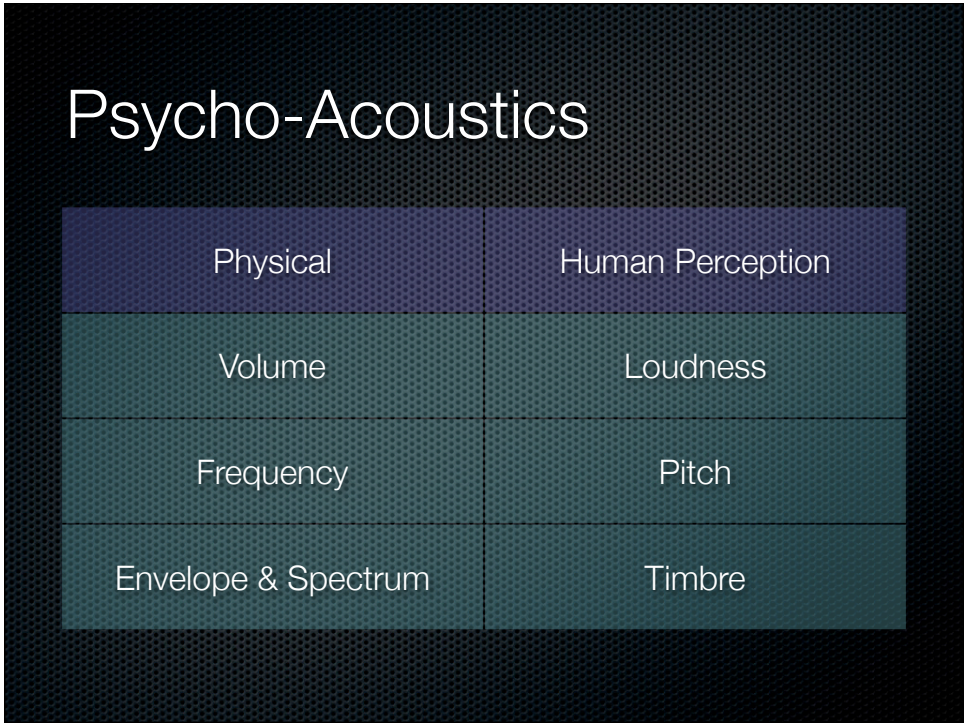
We just need to know a few things about waves...
1. a wave is a function in one dimension



Continuous means:
1. It is defined everywhere (it is has no “holes”)
2. Small changes in x -> small changes in y (it has no “jumps”)



Psycho-Acoustics is the study of human perception of sound.
It's relevant when designing audio effects, lossy compression schemes like MP3 and AAC, and at other times.

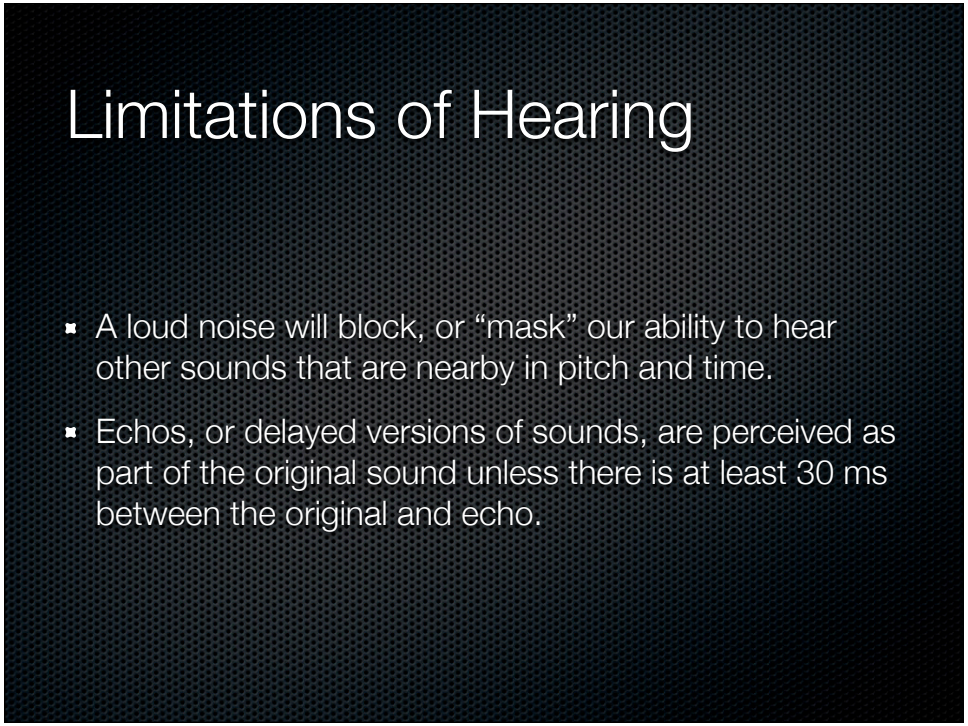


Many aspects of sound that we perceive, like loudness and pitch, correspond pretty closely to physical properties, like volume and frequency. Others like timber don’t correspond very closely at all.

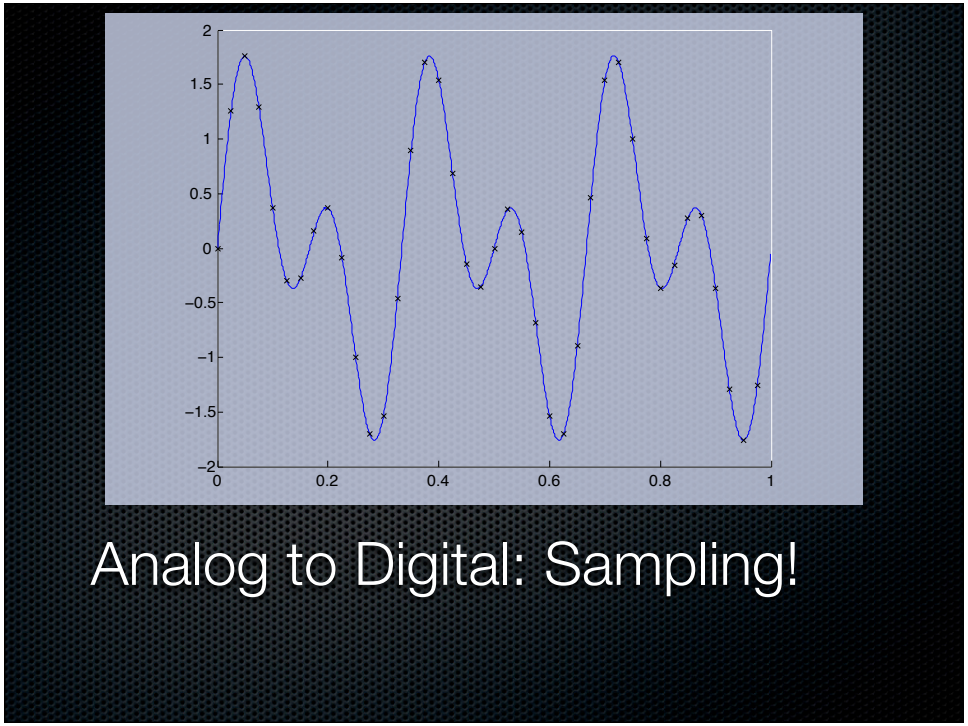
Limitations of Hearing

- Humans don't hear everything we can.
- We are very sensitive to changes in frequency (with about 1,400 individually discernible pitches in our range of hearing)
- We are not very sensitive to changes in volume (JND Volume is about 1 dB).
- The human ear can handle extremes: the loudest sound we are comfortable hearing is 1million x louder than the quietest sound we can hear.

JND, or "Just Noticeable Difference" is the smallest detectable difference between a first and a second level of a stimulus.



There are many other things that limit our hearing. These limitations are what allow lossy compression schemes to work.



Analog electrical signals (shown as a blue line) typically use voltage to represent some physical property, like air pressure. As air pressure goes up, voltage goes up in real time. In this way, voltage is analogous to the physical property or air pressure (hence the term analog).

To deal with signals digitally, we measure and record the amplitude of the analog signal at regular intervals. This gives us a stream of numbers, which is something the computer can deal with.

How often do we sample?

- Sample rates vary:
  - 8,000, 11,025, 16,000 Hz: common for speech/ voice applications.
  - 32,000 Hz: miniDV and other consumer applications
  - 44,100 Hz: “CD Quality” and most consumer music.
  - 48,000 Hz: Video.
  - Higher sample rates (up to 192,000 Hz) are available on pro sound cards.

By sampling more often, we can record higher frequency signals.

How are the samples formatted?

- All sorts of ways:
  - packed array of numbers (no headers).
  - may be interleaved or not (LRLRLR vs LLLRRR).
  - may be float or int.
  - Usually signed. Ints are 2's complement.
    - but for some reason windows often likes to give you signed ints when you are doing 8 bit audio
  - 12- and 20-bit audio is usually padded to 16- and 24-bit.

Format	Range	Use
8-bit Int “char” or “byte” or “int8_t”	-128 to 127 or 0 to 256	Old soundcards. Poor sound quality
16-bit Int “short” or “int16_t”	-32768 to 32767	Native format to most soundcards. CD Quality.
24-bit Int sometimes expanded to 32-bit int	-8388608 to 8388607	Pro soundcards. “pro” audio Quality.
32-bit Float “float”	-1 to 1	Convenient internal format for most computer-based DSP

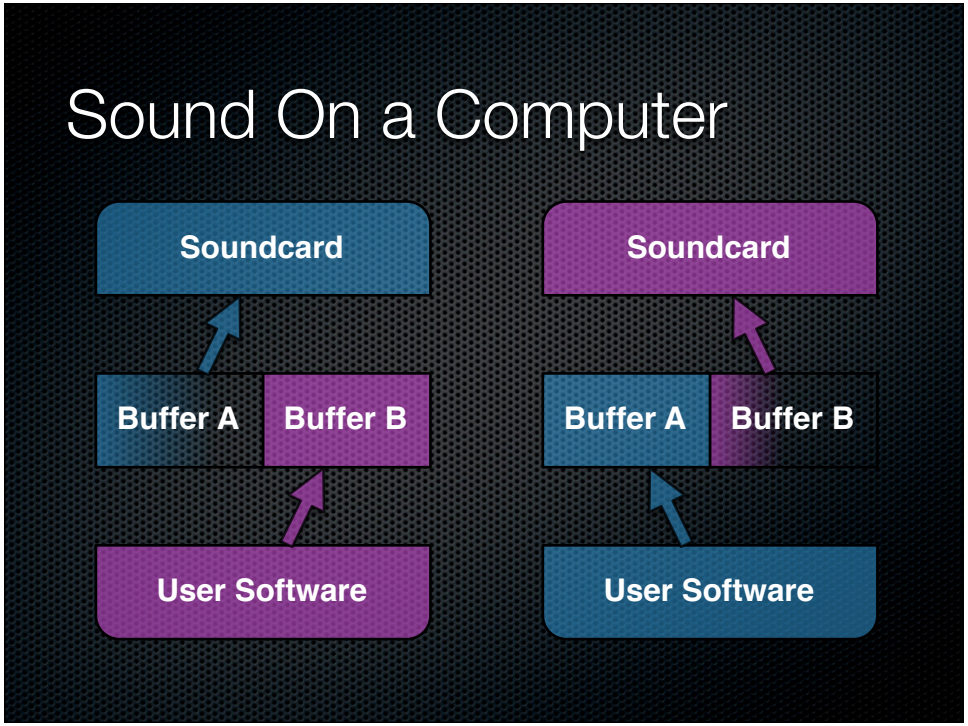
Some of the more common sample formats and their ranges.
Ints are typically what you get from your soundcard or sound file. Floats are typically what you work with.

How to convert from float to int:

- There's more than one right way, but there's lots of wrong ways. You won't go far off with this:
- (int value) / 2^{n-1} = float value
- (float value) * 2^{n-1} = int value
- where n is the number of bits.

Sound On a Computer

- Computers don't deal well with streams of numbers produced one sample at a time.
- We usually "buffer" the samples in small blocks of memory. Buffers sizes are often (but not always) powers of 2.
- The size of the buffer is important:
  - Smaller buffers means we can react to changes in user settings faster
  - Bigger buffers means more stable playback and recording

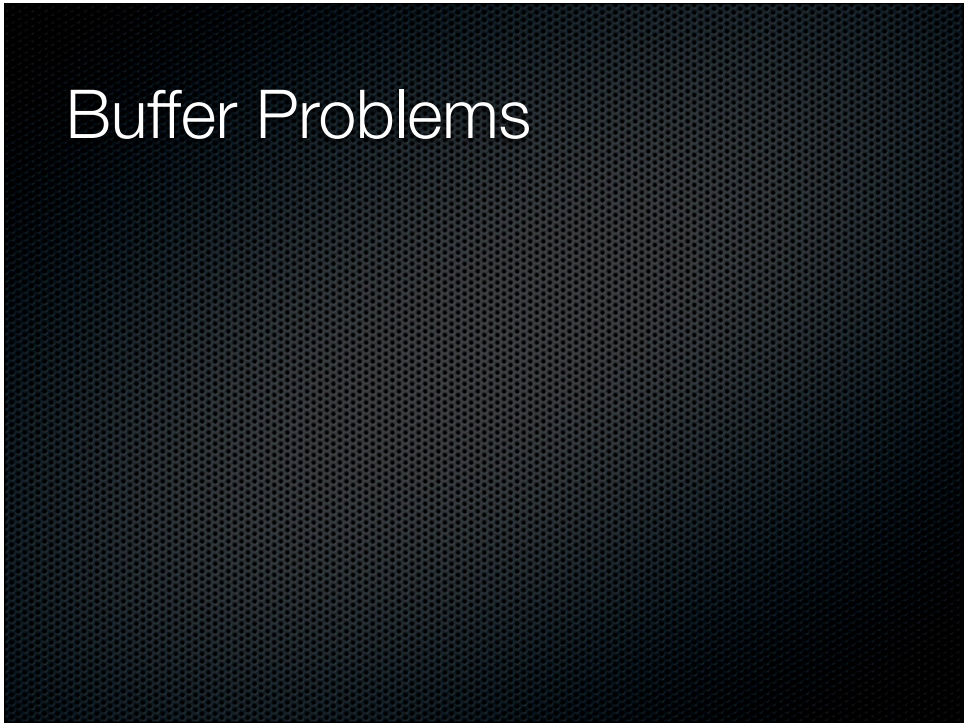


The simplest playback system: the soundcard reads one buffer while the software fills the second buffer. When the soundcard is done, it moves to the next buffer and the software switches to the first buffer.

The system can be made more elaborate with more buffers.

There are a variety of ways for communication to occur between the user software and the soundcard, including interrupt, poling, timers, and so on. The quality of these methods varies greatly, but AFAIK this is the basis for all modern soundcard drivers on modern OSes.

For this to work, the software must process buffers faster than the soundcard, every time, or a discontinuity may occur.



If buffers can’t be filled fast enough, we end up with sounds like this (buffers repeating and we start to hear discontinuities at the buffer boundaries)

Callback vs Blocking I/O

- Callback: software receives notification when a buffer is ready.
- Blocking: software reads and writes as it would to a file. If the sound hardware isn't ready, it forces the software to wait.
- Blocking I/O is usually a software layer written on top of native callbacks.

Callback vs Blocking I/O

- Generally speaking:
  - Callbacks are used in higher performance systems where latency (responsiveness) is more important.
  - blio is used where ease of programming is more important.

Callbacks vs Blocking

Callback	Blocking	Other
Windows: ASIO, WASAPI?	Windows: Direct Sound?	
Mac: Core Audio, Sound Manager		
Linux: ALSA, JACK?	*nix: OSS, ALSA	
Flash? Cinder? OF?	Java: Java Sound	OF? Javascript/ HTML5
PortAudio/rtAudio	PortAudio	

Both Blocking I/O and Callback are common. Some systems have other methods, which are higher level calls that allow playback and mixing, and sometimes other features like scheduling, volume and so on. These systems may be useful for simple applications or specialized applications like games, but generally don't allow direct, sample-level access to data.

Blocking I/O

```
//Complete, cross-platform example:
// Portaudio: test/patest_write_sine.c

main() {
   ...
   //Create the new stream:
   Stream stream( ... parameters ... );
   ...
   // loop: read/write data until done
   for( int i=0; i<whatever; ++i ) {
      stream.read( someData );
      ...
      stream.write( someOtherData );
      ...
   }
   //stop the stream:
   stream.stop();
}
```

Callback

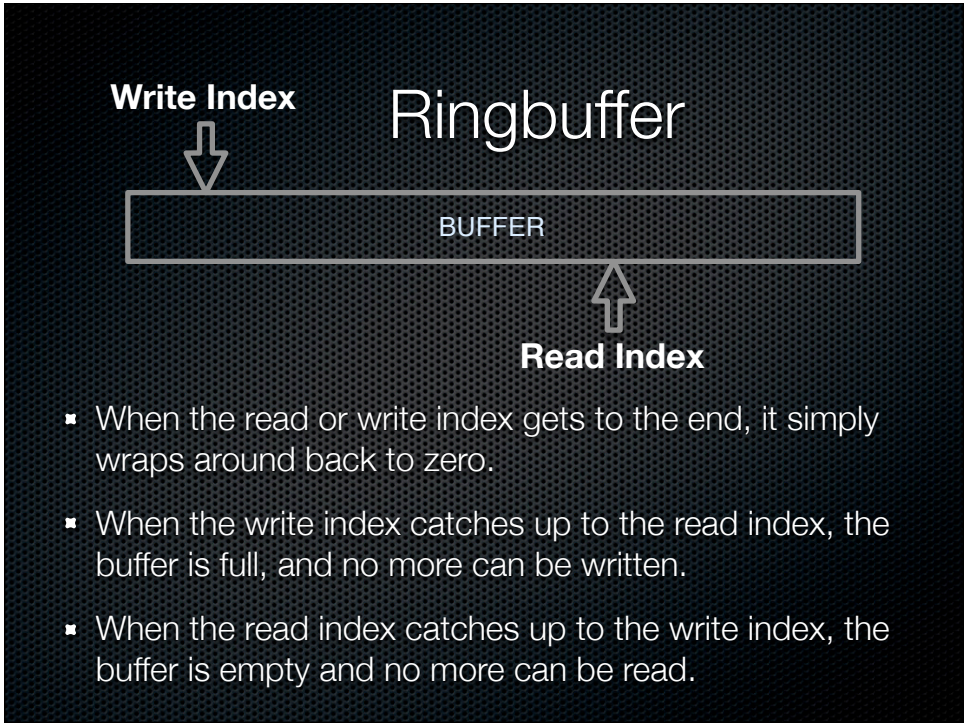
```
//for a complete, cross-platform example
//portaudio: test/patest_sine.c
boolean callback( ... );
main() {
   ...
   //Register your callback with the system:
   Stream stream( &callback, ... parameters ... );
   ...
   //start and stop the “stream” as needed, which
   // will cause the system to call the callback
   // whenever it needs audio.
   stream.start();
   while( streamIsRunning )
      sleep(10); //sleep, or whatever
   stream.close();
}
//Create a callback function:
boolean callback( void *audioIn, int sizeIn,
                  void *audioOut, int sizeOut) {
   //actual audio processing happens here!
   ...
   if( done )
      return true;
   else
      return false;
}
```

Callback doesn't seem so bad...

- The user-defined callback function must process audio and return in a prescribed amount of time.
- Specifically, the callback cannot:
  - perform I/O (disk or network I/O, terminal output, UI Updates, etc)
  - MUTEX lock (trylock may be okay)
  - new/malloc/free/delete
- Some systems place additional restrictions due to context.
- On some systems (Flash?) you can cheat.

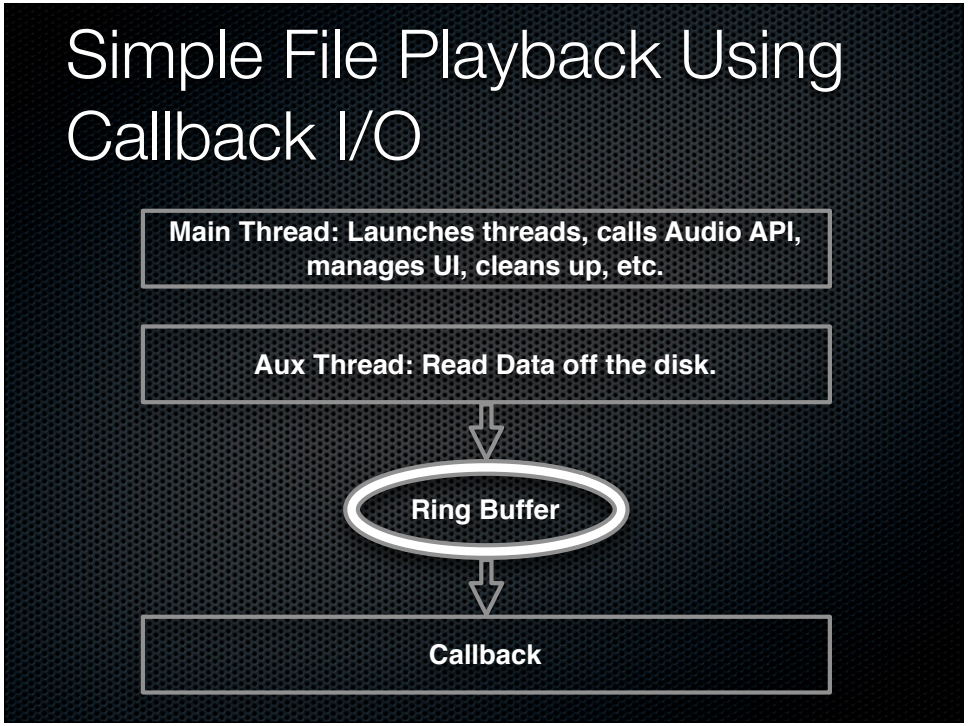
How do I get data into and out of my callback?

- Careful RT scheduling (Hard because most systems handle priority inversion poorly and have poor thread scheduling latency.)
- Lock-free/block-free data-structures and multiple threads. (Hard because C/C++ have terrible SMP multithreading support.)
- Simple, lock-free data-structures with memory-barriers for SMP safety.



For the ringbuffer to work correctly on SMP systems, you must apply memory barriers, sometimes called “fences.”

Simple File Playback Using Callback I/O



What else can I do?

- Use two copies of datastructures and a trylock.
- Other non-blocking datastructures exist, but not many in C/C++.
- Get excited about C++0x which will solve this stuff
- Use blio (blocking I/O)

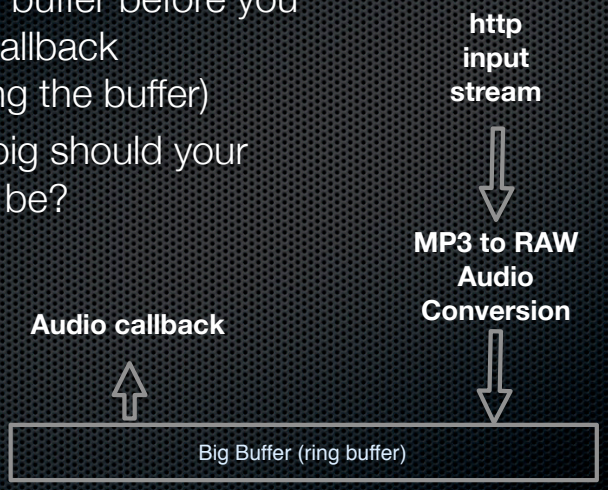
More on buffering

- If you are playing back audio from the intertubes, it will come at you in chunks.
- Moreover, the chunks you receive are often in the wrong format.
- This requires extra buffering.

Playing back from http

Fill the buffer before you start callback (priming the buffer)

How big should your buffer be?



Synchronization

- It's hard to change the speed of audio playback.
- Generally it's easy to change the speed of other things.
- So, synchronize playback of video and other things to audio, otherwise drift is inevitable, although usually small.

Audio clocks must be extremely stable or the sound quality suffers significantly, so most soundcards have crystal clocks built in. These clocks are ultimately the source of all audio timing. Trying to use something else as the source of timing is tricky to say the least.

Synchronizing on the web

- This is not an HTML class, but...
- HTML 5 allows you to register for updates to audio and video time playback.
- So you can have javascript do things when you get to a certain point in your audio/movie.
- Theoretically, you can use this to sync playback of multiple A/V events.
- You can't build a DAW this way. but you can do things like voiceovers. Maybe you can sync audio and video, I haven't tried.

Synchronizing on the web

```
<div id="stage">
  <video src="http://vid.ly/4f3q1f?content=video" controls></video>
  <div id="time"></div>
</div>
<script>
  (function(){
    var v = document.getElementsByTagName('video')[0]
    var t = document.getElementById('time');
    v.addEventListener('timeupdate',function(event){
      t.innerHTML = parseInt(v.currentTime) + ' - ' + v.currentTime;
    },false);
  })();
</script>
```

http://coding.smashingmagazine.com/2011/03/11/syncing-content-with-html5-video/

Synchronizing on the Web

- Javascript framework for web video with HTML 5:
  - popcornjs.org

DSP

- Reminder:
  - sound on a computer is a stream of numbers representing the amplitude at a given time.
  - Theoretically, that stream could go on forever. We deal with this by worrying about one buffer at a time.
  - We'll assume you have mono, floating point numbers.

we don't have time to even begin covering real DSP, but we can cover some basics.

Volume

- To adjust the volume of the a signal, simply multiply each sample in the signal by a constant.
- >1 to increase volume
- <1 to decrease volume
- to convert from dB: $10^{x/20}$, where x is the dB value.
  - eg. gain of -6 dB = $10^{-6/20} = 10^{-0.3} \approx .501$

Volume

```
adjustVolume( float audio[], float gain ) {
  for( int i=0; i<audio.length; ++i )
    audio[i] *= gain;
}
// adjust volume linearly, which is not always how we want to do it,
// but at least it's smooth (no discontinuities)

//For more on linear interpolation:
// http://blog.bjornroche.com/2010/10/
// linear-interpolation-for-audio-in-c-c.html

adjustVolume( float audio[], float gainStart, float gainEnd ) {
  for( int i=0; i<audio.length; ++i ) {
    // gain is the weighted average of the start and end gain
    float weight=i/audio.length
    float gain=(1-weight)*gainStart + weight*gainEnd;
    audio[i] *= gain ;
  }
}
```

To Deal With “Overs”

- There’s more than one way to deal with “overs,” or out of range values. The right way depends on context.
- The simplest way is with hard “clipping.”
- This creates distortion, but sometimes it’s all you can do.
- Your Audio API may do this for you.

```
clip( float audio[] ) {
  for( int i=0; i<audio.length; ++i )
    if( audio[i] > 1 )
      audio[i] = 1;
    else if( audio[i] < -1 )
      audio[i] = -1;
}
```

Mixing

To mix two or more signals, just add them together, sample, by sample. (This is super-position)

```
mix( float track1[], float track2[], float output[] ) {
  for( int i=0; i<track1.length; ++i )
    output[i] = track1[i] + track2[i] ;
}
```

watch out for clipping, here, too!

Panning

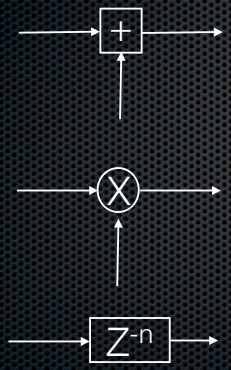
- There are different ways to pan.
- Here is one

```
// 0 < panValue < 1
// 0 ~ left
// 1 ~ right
// .5 ~ center

pan( float in[], float left[], float right[], float panValue ) {
    for( int i=0; i<in.length; ++i ) {
      l[i]=sqrt(panValue)*in[i];
      r[i]=sqrt(1-panValue)*in[i];
    }
}
```

Other DSP

- Other DSP is made up of a number of components, including:



Addition
(Usually 2 or more signals)

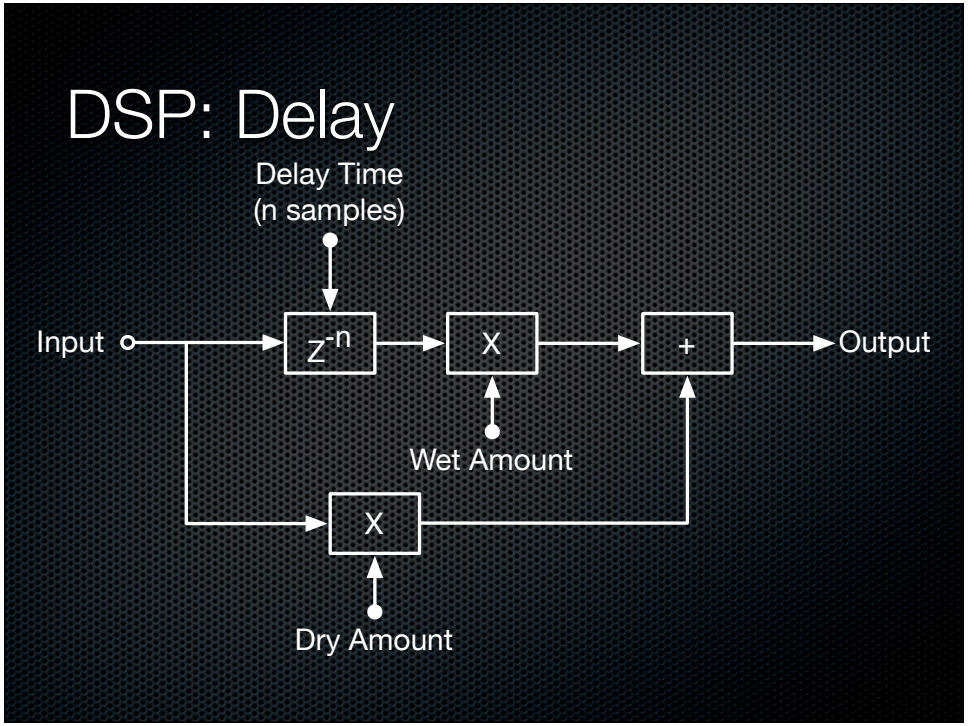
Multiplication
(Usually by a constant
or a control signal)

Delay by n samples

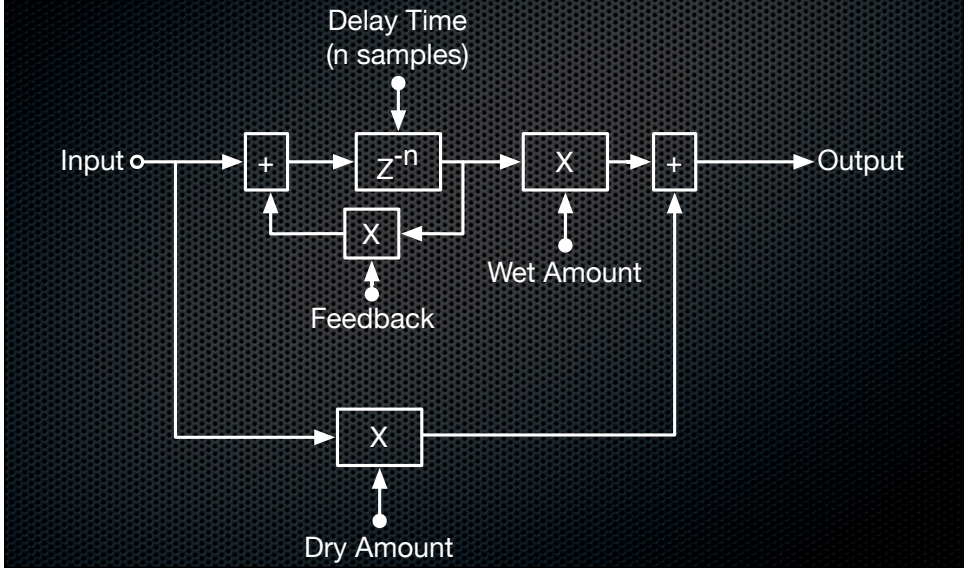
DSP (LSI)

- With just multiplication by a constant, addition, and delay, we can make any LSI (Linear Shift Invariant) effect.
- What's a LSI effect?
  - reverb
  - delay
  - EQ

DSP: Delay

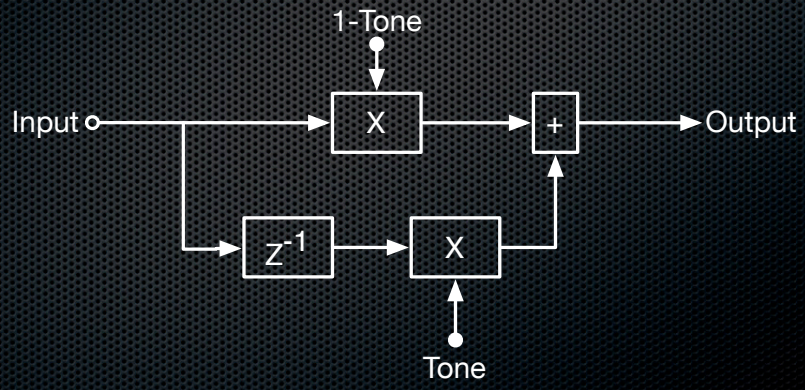

Delay Time
(n samples)
Input
z^{-n}
X
+
Output
Wet Amount
X
Dry Amount

DSP: Delay with Feedback



Non-Recursive (FIR) EQ

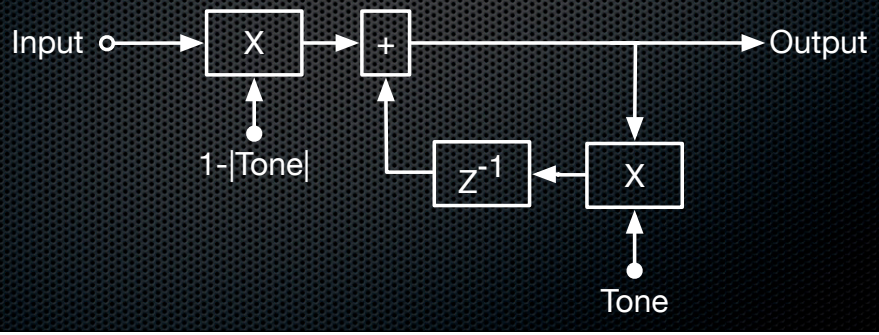
- When tone > 0, you can think of this as a moving average filter. Thus, it eliminates high frequencies, and keeps low ones.
- for tone < 0, it has the opposite effect.



This filter is useful in reverbs and simple tone controls, but you won't usually find it in the "EQ" section of your audio software.

Recursive (IIR) EQ

- This filter is more selective, but…
- It is unstable when tone > 1 or tone < -1
- errors in calculation accumulate as the each new value depends on every prior value.
- “Phase” gets shifted (whatever the heck that means).



This filter does a better job of picking out high and low frequencies, but there are problems with it.

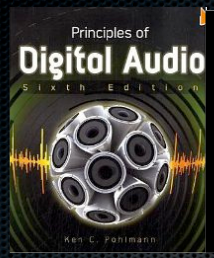
Other FX

- Other basic effects you might want to learn about include:
  - Reverb
  - Higher order EQ
  - Compression/Limiting/Gating
  - Pitch shifting/time stretching
  - Chorus/Flanging
  - Phasing

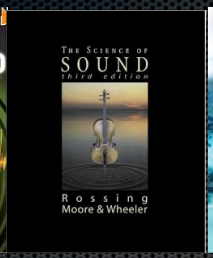
Other FX

- Some APIs have effects built-in, so you don't have to reinvent the wheel.
- Don't expect one API's effects, no matter how simple, to sound or behave like another's.

Resources

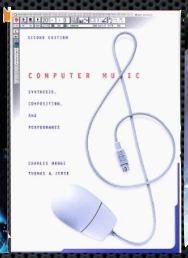


Priciples of Digital Audio Pohlmann

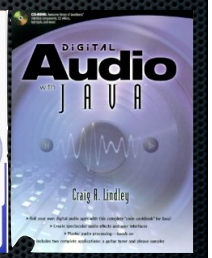


The Science of Sound Rossing

DAFX Zölzer



Computer Music Dodge & Jerse



Digital Audio with Java Lindley

Resources

- portaudio.com
- popcornjs.org
- http://coding.smashingmagazine.com/2011/03/11/syncing-content-with-html5-video/
- musicdsp.org
  - http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

Resources: Languages and Frameworks

- CSound http://www.csounds.com/
- Processing http://processing.org/
- Matlab http://www.mathworks.com/products/matlab/index.html
- Octave http://www.gnu.org/software/octave/
- Scilab http://www.scilab.org/
- OpenFrameworks http://www.openframeworks.cc/
- Cinder http://libcinder.org/

Resources: C-based I/O

- PortAudio http://www.portaudio.com/
- RTAudio http://www.music.mcgill.ca/~gary/rtaudio/
- libsndfile http://www.mega-nerd.com/libsndfile/
- Secret Rabit Code http://www.mega-nerd.com/SRC/

Resources: Me

- My Blog: http://blog.bjornroche.com
- http://xoaudio.com
- http://xonami.com
- I don’t tweet, but you can find me on linkedin.