CS 1124Media Computation

Oct 6, 2008

Steve Harrison

TODAY

- Midterm
- Introduction to working with sound
- HW 5 - Faster and Faster

TODAY

- Midterm
- Introduction to working with sound
- HW 5 - Faster and Faster

MID TERM

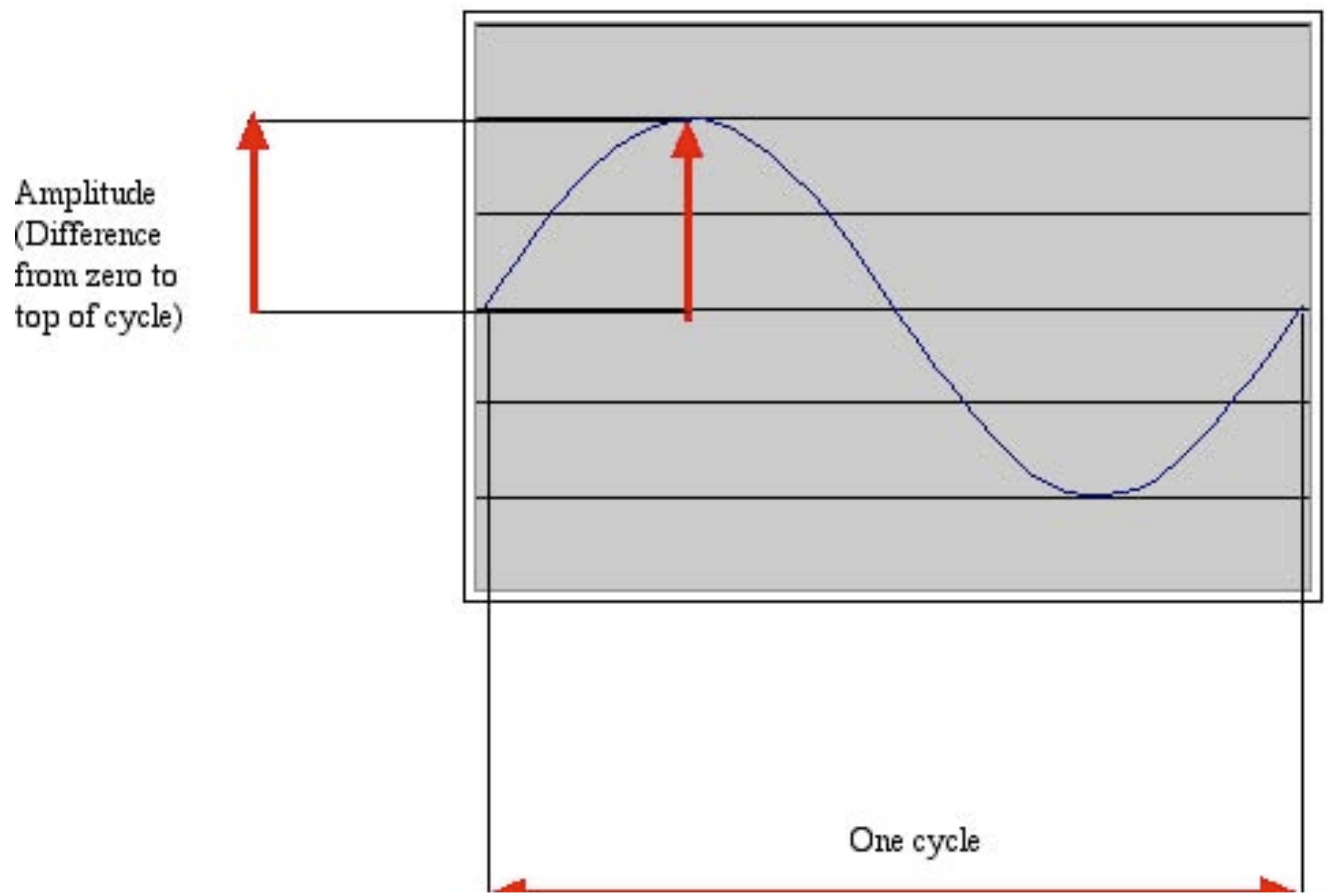
- Still being graded...
- One "gotcha":
  - in gray-scale to posterized question - first range was <85, second range was >85 thus if == 85, THEREFORE SHOULD BE YELLOW

TODAY

- Midterm
- Introduction to working with sound
- HW 5 - Faster and Faster

How sound works: Acoustics, the physics of sound

- Sounds are waves of air pressure
  - **Sound comes in cycles**
  - **The frequency of a wave is the number of cycles per second (cps), or Hertz**
    - (Complex sounds have more than one frequency in them.)
  - **The amplitude is the maximum height of the wave**



Volume and pitch: Psychoacoustics, the psychology of sound

- Our perception of volume is related (logarithmically) to changes in amplitude
  - **If the amplitude doubles, it's about a 3 decibel (dB) change**
- Our perception of pitch is related (logarithmically) to changes in frequency
  - **Higher frequencies are perceived as higher pitches**
  - **We can hear between 20 Hz and 20,000 Hz (20 kHz)**
  - **A above middle C is 440 Hz**

ERROR in the book!

“Logarithmically?”

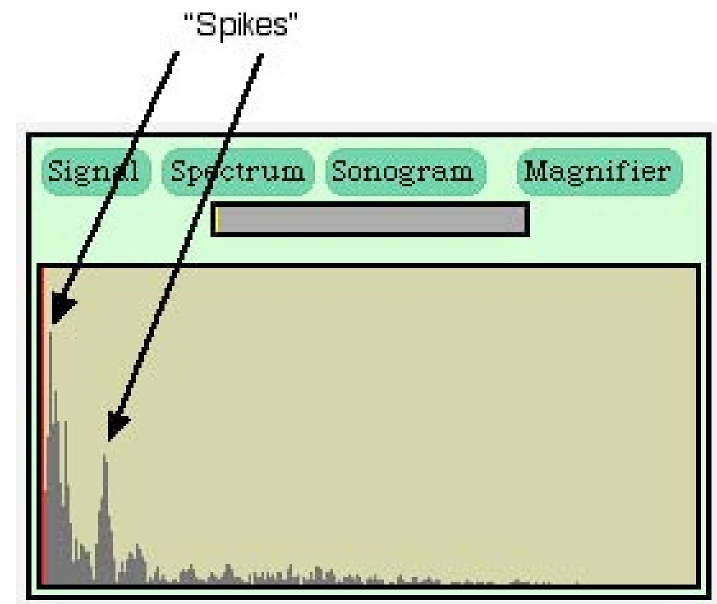
- It’s strange, but our hearing works on *ratios* not *differences*, e.g., for pitch.
  - **We hear the difference between 200 Hz and 400 Hz, as the same as 500 Hz and 1000 Hz**
  - **Similarly, 200 Hz to 600 Hz, and 1000 Hz to 3000 Hz**
- Intensity (volume) is measured as *watts per meter squared*
  - **A change from 0.1W/m^2 to 0.01 W/m^2, sounds the same to us as 0.001W/m^2 to 0.0001W/m^2**

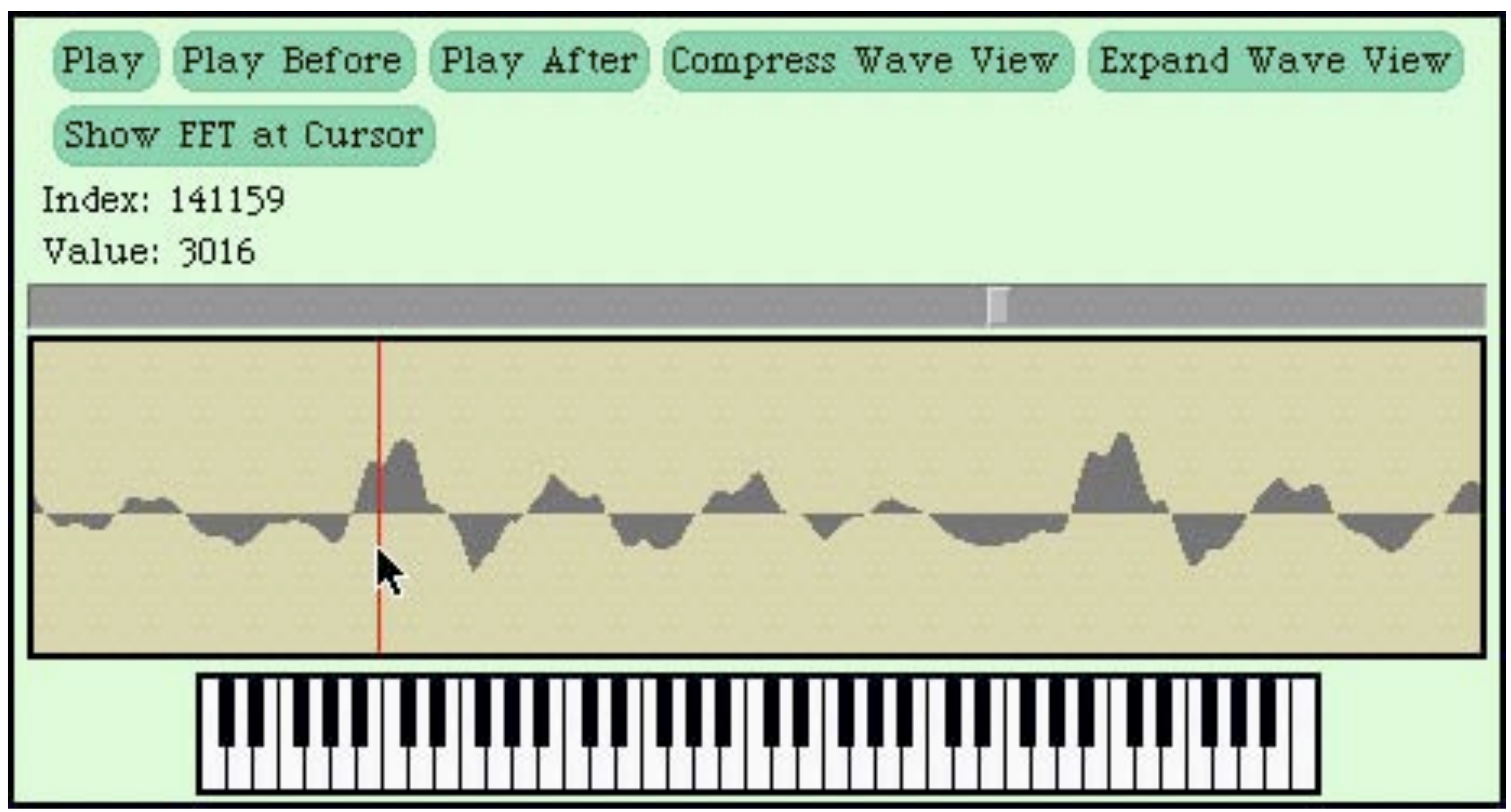
Decibel is a logarithmic measure

- A *decibel* is a ratio between two intensities: $10 * \log_{10}(I_1/I_2)$
  - **As an absolute measure, it's in comparison to threshold of audibility**
  - **0 dB can't be heard.**
  - **Normal speech is 60 dB.**
  - **A shout is about 80 dB**

Demonstrating Sound MediaTools

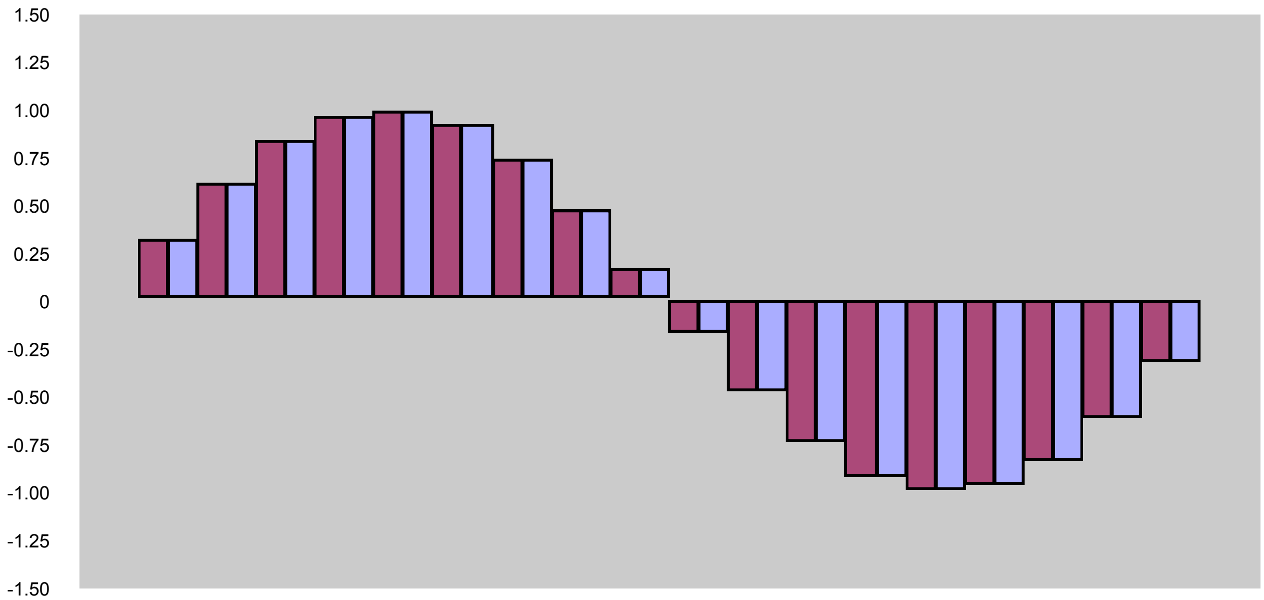
Fourier transform (FFT)





Digitizing Sound: How do we get that into numbers?

- Remember in calculus, estimating the curve by creating rectangles?
- We can do the same to estimate the sound curve
  - **Analog-to-digital conversion (ADC) will give us the amplitude at an instant as a number: a sample**
  - **How many samples do we need?**



Nyquist Theorem

- We need twice as many samples as the maximum frequency in order to represent (and recreate, later) the original sound.
- The number of samples recorded per second is the *sampling rate*
  - **If we capture 8000 samples per second, the highest frequency we can capture is 4000 Hz**
    - That's how phones work
  - **If we capture more than 44,000 samples per second, we capture everything that we can hear (max 22,000 Hz)**
    - CD quality is 44,100 samples per second

Digitizing sound in the computer

- Each sample is stored as a number (two bytes)
  - called a "word" ← Not in the book
- What's the range of available combinations?
  - 16 bits, 2^{16} = 65,536
  - But we want both positive and negative values
    - To indicate compressions and rarefactions.
  - What if we use one bit to indicate positive (0) or negative (1)?
  - That leaves us with 15 bits
  - 15 bits, 2^{15} = 32,768
  - One of those combinations will stand for zero
    - We'll use a "positive" one, so that's one less pattern for positives

+/- 32K (32,767)

- Each sample can be between -32,768 and 32,767

Why such a bizarre number?

Because 32,768 + 32,767 + 1 = 2^{16}

< 0

> 0

0

i.e. 16 bits, or 2 bytes

Compare this to 0 ... 255 for light intensity

(i.e. 8 bits or 1 byte)

bytes, words and binary numbers

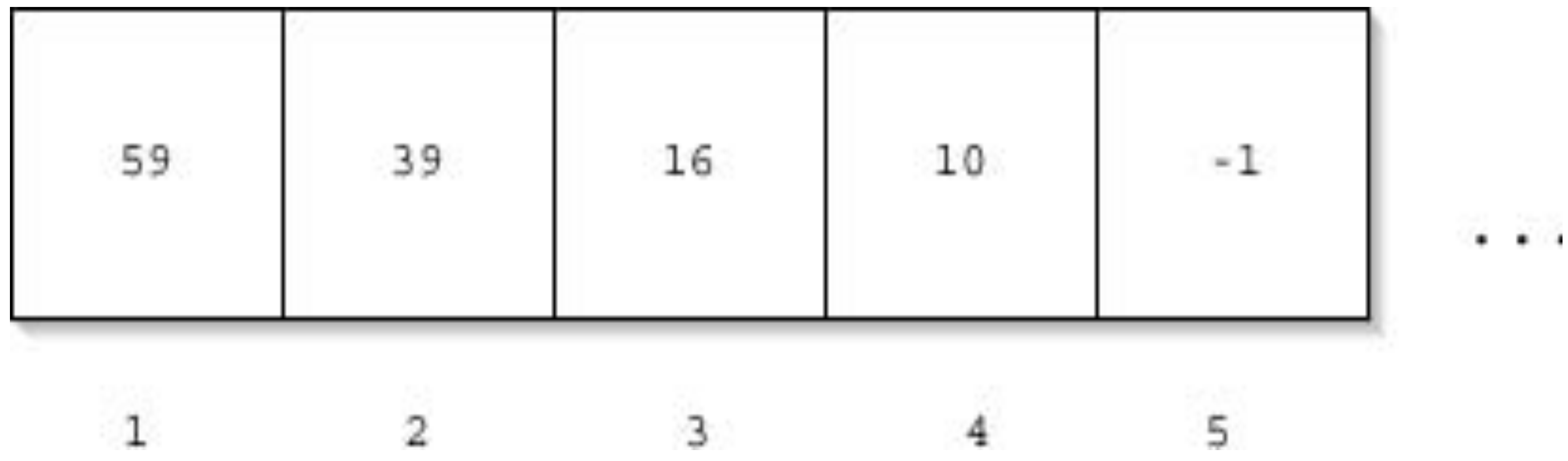
- Each sample is stored as a number (two bytes)
  - **called a "word"** ← Not in the book
- What's the range of available combinations?
  - **16 bits, 2^{16} = 65,536 combinations**
  - **or -32,768 to 32,767**
  - **or <u>(two's complement arithmetic)</u>**

+/-	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**32,767**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	**2**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**-1**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	**-2**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**-32,768**

Sounds as arrays

- Samples are just stored one right after the other in the computer's memory
- That's called an *array*
  - **It's an especially efficient (quickly accessed) memory structure**
  - **each sample is two bytes (or ONE WORD)**

(Like pixels in a picture)



Working with sounds

- We'll use **pickAFile** and **makeSound**.
  - **We want .wav files**
- We'll use **getSamples** to get all the *sample objects* out of a sound
- We can also get the value at any index with **getSampleValueAt**
- Sounds also know their length (**getLength**) and their sampling rate (**getSamplingRate**)
- Can save sounds with **writeSoundTo(sound,"file.wav")**

Demonstrating Working with Sound in JES

```
>>> filename = pickAFile()
>>> print filename
/Users/guzdial/mediasources/preamble.wav
>>> sound = makeSound(filename)
>>> print sound
Sound of length 421109
>>> samples = getSamples(sound)
>>> print samples
Samples, length 421109
>>> print getSampleValueAt(sound, 1)
36
>>> print getSampleValueAt(sound, 2)
29
```

Demonstrating working with samples

```
>>> print getLength(sound)
220568
>>> print getSamplingRate(sound)
22050.0
>>> print getSampleValueAt(sound, 220568)
68
>>> print getSampleValueAt(sound, 220570)
I wasn't able to do what you wanted.
The error java.lang.ArrayIndexOutOfBoundsException has occured
Please check line 0 of
>>> print getSampleValueAt(sound, 1)
36
>>> setSampleValueAt(sound,1, 12)
>>> print getSampleValueAt(sound, 1)
12
```

Working with Samples

- We can get sample objects out of a sound with **getSamples(sound)** or **getSampleObjectAt(sound, index)**
- A sample object remembers its sound, so if you change the sample object, the sound gets changed.
- Sample objects understand **getSample(sample)** and **setSample(sample, value)**

Example: Manipulating Samples

```
>>> soundfile=pickAFile()
>>> sound=makeSound(soundfile)
>>> sample=getSampleObjectAt(sound, 1)
>>> print sample
Sample at 1 value at 59
>>> print sound
Sound of length 387573
>>> print getSound(sample)
Sound of length 387573
>>> print getSample(sample)
59
>>> setSample(sample, 29)
>>> print getSample(sample)
29
```

“But there are thousands of these samples!”

- How do we do something to these samples to manipulate them, when there are thousands of them per second?
- We use a *loop* and get the computer to *iterate* in order to do something to each sample.
- An example loop:

```
for sample in getSamples(sound):
    value = getSample(sample)
    setSample(sample, value)
```

Let’s try a few things …

- normalize(sound)
  - from the book
  - and revised with abs(), testing for largest @ limit of 32,767 or -32,768 and return sound
- double(sound)
  - what happens if > 32,767?
  - what does it sound like? what does it look like?

Normalizing

- A few ways to think about “normalizing”:
  - **use the whole enchilada (don’t waste any bits…)**
  - **make everything use the same scale (0 to 100%)**
  - **need 2 loops -- one to find largest and one to reset**

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, getSample(sample) )
  multiplier = 32767.0 / largest
  print “Largest”, largest, “multiplier is”, multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
```

Normalizing (modified)

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, abs( getSample(sample) ) )
    if largest > 32766 :
        return sound
  multiplier = 32768.0 / largest
  print “Largest”, largest, “multiplier is”, multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
  return sound
```

Normalizing (modified)

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, abs( getSample(sample) ) )
    if largest > 32766 :
        return sound
  multiplier = 32768.0 / largest
  print "Largest", largest, "multiplier is", multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
  return sound
```

Normalizing (modified)

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, abs( getSample(sample) ) )
    if largest > 32766 :
        return sound
  multiplier = 32768.0 / largest
  print “Largest”, largest, “multiplier is”, multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
  return sound
```

Normalizing (modified)

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, abs( getSample(sample) ) )
    if largest > 32766 :
        return sound
  multiplier = 32768.0 / largest
  print “Largest”, largest, “multiplier is”, multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
  return sound
```

Normalizing (modified)

```
def normalize( sound ) :
  largest = 0
  for sample in getSamples(sound):
    largest = max( largest, abs( getSample(sample) ) )
    if largest > 32766 :
        return sound
  multiplier = 32768.0 / largest
  print “Largest”, largest, “multiplier is”, multiplier
  for sample in getSamples(sound):
    setSample(sample, getSample(sample) * multiplier)
  return sound
```

Doubling the amplitude

```
def double( sound ) :
  for sample in getSamples(sound):
    value = getSample(sample)
    setSample(sample, value * 2)
```

TODAY

- Midterm
- Introduction to working with sound
- HW 5 - Faster and Faster

Assignment 5 - Due Wed 10/15

- Faster and Faster (or Higher and Higher)
- For a sound:
  - **increasingly compress the sound:**
    - 0% - 25%   1:1 (no compression)
    - 25%-50%   1:1.25
    - 50% - 75% 1:1.5
    - 75% -100% 1:2 (twice as fast)
  - **print out how much shorter in seconds the compressed sound is**
  - **save the sound to a file**

Assignment 5

- For extra credit on Final Exam
- For a sound:
  - **#comment that you are doing the challenge**
  - **increasingly compress the sound:**
    - 0% - 25%   1:1 (no compression)
    - 25% -100% smoothly change from 1:1 to 1:2 (twice as fast) instead of in steps
  - **print out how much shorter in seconds the compressed sound is**
    - this method should produce different results from basic
  - **save the sound to a file**

Questions?

TODAY

- Midterm
- Introduction to working with sound
- HW 5 - Faster and Faster

Coming attractions

- Today - LAST DAY TO REGISTER TO VOTE
- For Next Monday:
  - **read Chapter 7**
  - **Quiz 7 due 10:00 am**