Sound Encoding and Manipulation

What is sound?

• Waves of air (or water if underwater) pressure
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

Decibel is a logarithmic measure

- A **decibel** is a ratio between two intensities: $10 \times \log_{10}(I_1/I_2)$
  - 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
Intensity of Sound

- Decibel Scale

<table>
<thead>
<tr>
<th>Decibel Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>(dB)</td>
</tr>
<tr>
<td>Threshold of pain</td>
</tr>
<tr>
<td>Highway traffic at 30 m</td>
</tr>
<tr>
<td>Residential area at night</td>
</tr>
<tr>
<td>Quiet restaurant</td>
</tr>
<tr>
<td>Rustling of leaves</td>
</tr>
<tr>
<td>Threshold of hearing</td>
</tr>
</tbody>
</table>

Source: N. HWA

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 5 Hz and 20,000 Hz (20 kHz)
  - Middle C is 262 Hz
- 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
Demonstrating Sound MediaTools

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)
- 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

- 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 i.e. 16 bits, or 2 bytes

Compare this to 0..255 for light intensity
(i.e. 8 bits or 1 byte)

Sounds as arrays

- Samples are just stored one right after the other in the computer’s memory
- That’s called an array (Like pixels in a picture)
  - It’s an especially efficient (quickly accessed) memory structure
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

```python
>>> filename = pickAFile()
>>> print filename
c:\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

```python
>>> 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 occurred
Please check line 0 of
>>> print getSampleValueAt(sound, 1)
36
>>> setSampleValueAt(sound,1, 12)
>>> print getSampleValueAt(sound, 1)
12
```
Example: Manipulating Samples

```python
>>> 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
```

Can you hear the difference?

“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:

```python
    for sample in getSamples(sound):
        value = getSample(sample)
        setSample(sample, value)
```
Recipe to Increase the Volume

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

Using it:
```python
>>> setMediaPath()
>>> s = makeSound("gettysburg10.wav")
>>> increaseVolume(s)
>>> play(s)
>>> writeSoundTo(s, "louder-g10.wav")
```

Starting the loop

- `getSamples(sound)` returns a sequence of all the sample objects in the `sound`.
- The `for` loop makes `sample` be the first sample as the block is started.

Need to use `getSample` to get the actual value.
Executing the block

- We get the value of the sample named `sample`.
- We set the value of the sample to be the current value (variable `value`) times 2.

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

Next sample

- Back to the top of the loop, and `sample` will now be the second sample in the sequence.

```python
def increaseVolume(sound):
    for sample in getSamples(sound):
        value = getSample(sample)
        setSample(sample, value * 2)
```
And increase that next sample

- We set the value of this sample to be the current value (variable value) times 2.

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

And on through the sequence

- The loop keeps repeating until all the samples are doubled

```python
def increaseVolume(sound):
    for sample in getSamples(sound):
        value = getSample(sample)
        setSample(sample, value * 2)
```
How are we *sure* that that worked?

```python
>>> print s
Sound of length 220567
>>> print f
C:\mediasources\gettysburg10.wav
>>> soriginal = makeSound(f)
>>> print getSampleValueAt(s, 1)
118
>>> print getSampleValueAt(soriginal, 1)
59
>>> print getSampleValueAt(s, 2)
78
>>> print getSampleValueAt(soriginal, 2)
39
>>> print getSampleValueAt(s, 1000)
-80
>>> print getSampleValueAt(soriginal, 1000)
-40
```

Decreasing the volume

```python
def decreaseVolume(sound):
    for sample in getSamples(sound):
        value = getSample(sample)
        setSample(sample, value * 0.5)
```

This works *just* like `increaseVolume`, but we’re *lowering* each sample by 50% instead of doubling it.
Recognize some similarities?

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

def decreaseVolume(sound):
    for sample in getSamples(sound):
        value = getSample(sample)
        setSample(sample, value * 0.5)

def increaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p, value*1.2)

def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p, value*0.5)
```

Does increasing the volume change the volume setting?

- No
  - The physical volume setting indicates an upper bound, the potential loudest sound.
  - Within that potential, sounds can be louder or softer
    - They can fill that space, but might not
    - E.g. TV commercial volume louder than show

- What happens if we keep calling increaseVolume over and over again?
**Avoiding clipping**

- Why are we being so careful to stay within range? What if we just multiplied all the samples by some big number and let some of them go over 32,767?
- The result then is *clipping*
  - Clipping: The awful, buzzing noise whenever the sound volume is beyond the maximum that your sound system can handle.

**Maximizing volume**

- How, then, do we get maximal volume?
  - (e.g. automatic recording level)
- It’s a three-step process:
  1. find the current loudest value (largest sample).
  2. find how much we can increase/decrease that value to fill the available space
     - We want to find the amplification factor $amp$, where $amp \times loudest = 32767$
     - In other words: $amp = 32767/loudest$
  3. amplify each sample by multiplying it by $amp$
Maxing (*normalizing*) the sound

```python
def normalize(sound):
    largest = 0
    for s in getSamples(sound):
        largest = max(largest, getSample(s))
    amplification = 32767.0 / largest

    print("Largest sample value in original sound was", largest)
    print("Amplification multiplier is", amplification)

    for s in getSamples(sound):
        louder = amplification * getSample(s)
        setSample(s, louder)
```

.max()

- **max()** is a function that takes *any* number of parameters, and returns the largest.
- There is also a function **min()** which works similarly but returns the minimum.
- We could also write these ourselves, like we did on homework #2

```python
>>> print max(1, 2, 3)
3
>>> print max(4, 67, 98, -1)
98
```
Or: use if instead of max

```python
def normalize(sound):
    largest = 0
    for s in getSamples(sound):
        if getSample(s) > largest:
            largest = getSample(s)
    amplification = 32767.0 / largest
    print "Largest sample value in original sound was", largest
    print "Amplification factor is", amplification
    for s in getSamples(sound):
        louder = amplification * getSample(s)
        setSample(s, louder)
```

Aside: positive and negative extremes assumed to be equal

- We're making an assumption here that the maximum positive value is also the maximum negative value.
  - That should be true for the sounds we deal with, but isn't necessarily true
- Try adding a constant to every sample.
  - That makes it non-cyclic
    - I.e. the compressions and rarefactions in the sound wave are not equal
    - But it's fairly subtle what's happening to the sound.
Processing only *part* of the sound

- What if we wanted to increase or decrease the volume of only *part* of the sound?
- Q: How would we do it?
- A: We’d have to use a `range()` function with our `for` loop
  - Just like when we manipulated only part of a picture by using `range()` in conjunction with `getPixels()`
  - More about this next time....