

CS1315: Introduction to Media Computation

Sound (2)

Sound volume

Sound manipulations

Advanced Sound

- Increasing and decreasing volume. Clipping.
- Splicing sounds and echo effects.
- Processing parts of sounds. Sample indexing.
- Playing sounds backward. Copying samples.
- Changing pitch of sounds. Sampling.

Decreasing the volume

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

- Try this with any sound.
- Use *openSoundTool* from the command pane to play a sound and view its waveform.

Recipe to Increase the Volume

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

Turn your computer's volume down before playing this!

Recognize some similarities?

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

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

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

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

But what if the sound now has $|value| > 32k$?

Maximizing volume

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

Maxing (*normalizing*) the sound

```
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)
```

This loop finds the loudest value

Note: Real, not integer division

This loop actually amplifies the sound

max()

- **max()** is a function that takes *any* number parameters, and returns the largest.
- There is also a function **min()** which works similarly but returns the minimum

Or: use **if** instead of **max**

```
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)
```

Check each in turn to see if it's the largest so far

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.

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.

Making more complex sounds

- We know that natural sounds are often the combination of multiple sounds.
- Adding waves is easy. Simply add the samples at the same index in the two waves:
 - You may need to normalize both first to avoid clipping
 - Or, instead of adding the sounds, you can average them

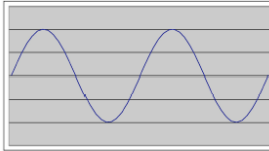
```
for srcSample in range(1, getLength(source)+1):
    destValue = getSampleValueAt(dest, srcSample)
    srcValue = getSampleValueAt(source, srcSample)
    setSampleValueAt(source, srcSample, srcValue+destValue)
```

Adding sounds

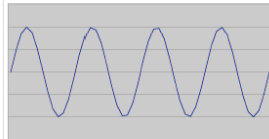
The first two are sine waves generated in Excel.

The third is just the sum of the first two columns.

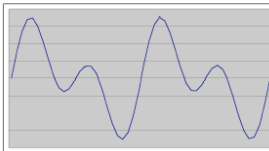
a



b



a + b = c



Uses for adding sounds

- We can mix sounds
 - We even know how to change the volumes of the two sounds, even over time (e.g., fading in or fading out)
- We can create echoes
 - Just add a sound to itself after a delay
 - Or, do this multiple times, with the later copies at reduced amplitude
- We can add sine (or other) waves together to create kinds of instruments/sounds that do not physically exist, but which sound interesting and complex

A function for adding two sounds

```
def addSoundInto(sound1, sound2):
```

```
    for sampleNmr in range(1, getLength(sound1)+1):
        sample1 = getSampleValueAt(sound1, sampleNmr)
        sample2 = getSampleValueAt(sound2, sampleNmr)
        setSampleValueAt(sound2, sampleNmr, sample1 + sample2)
```

Notice that this adds sound1 and sound2 by adding sound1 *into* sound2

Adding sounds with a delay

```
def makeChord(sound1, sound2, sound3):
    for index in range(1, getLength(sound1)):
        s1Sample = getSampleValueAt(sound1, index)
        if index > 1000:
            s2Sample = getSampleValueAt(sound2, index - 1000)
            setSampleValueAt(sound1, index, s1Sample + s2Sample)
        if index > 2000:
            s3Sample = getSampleValueAt(sound3, index - 2000)
            setSampleValueAt(sound1, index, s1Sample + s2Sample + s3Sample)
```

-Add in sound2 after 1000 samples

-Add in sound3 after 2000 samples

Note that in this version we're adding into sound1!

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()`

Knowing where we are in the sound

- More complex operations require us to know where we are in the sound, which sample
 - Not just process all the samples exactly the same
- Examples:
 - Reversing a sound
 - It's just copying amplitude from one location in the sound to another
 - Changing the frequency of a sound
 - Copying every nth sample compresses the wave & so increases pitch
 - Splicing sounds
 - Adding sounds to a sound "canvas"

Increasing volume by *sample index*

```
def increaseVolumeByRange(sound):  
    for sampleNumber in range(1, getLength(sound) + 1):  
        value = getSampleValueAt(sound, sampleNumber)  
        setSampleValueAt(sound, sampleNumber, value * 2)
```

This really is the same as:

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

Recipe to play a sound backwards (Trace it!)

```
def playBackward(filename):  
    source = makeSound(filename)  
    dest = makeSound(filename)  
  
    srcIndex = getLength(source)  
    for destIndex in range(1, getLength(dest) + 1):  
        srcSample = getSampleValueAt(source, srcIndex)  
        setSampleValueAt(dest, destIndex, srcSample)  
        srcIndex = srcIndex - 1  
  
    return dest
```

Start at end of sound

Work backward

Return the processed sound for further use in the function that calls playBackward

How does this work?

- We make two copies of the sound
- The **srcIndex** starts at the end, and the **destIndex** goes from 1 to the end.
- Each time through the loop, we copy the sample value from the **srcIndex** to the **destIndex**

Note that the **destIndex** is *increasing* by 1 each time through the loop, but **srcIndex** is *decreasing* by 1 each time through the loop

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)

    srcIndex = getLength(source)
    for destIndex in range(1, getLength(dest) + 1):
        srcSample = getSampleValueAt(source, srcIndex)
        setSampleValueAt(dest, destIndex, srcSample)
        srcIndex = srcIndex - 1

    return dest
```

Starting out (3 samples here)

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

← You are here

```
srcIndex = getLength(source)
for destIndex in range(1, getLength(dest) + 1):
    srcSample = getSampleValueAt(source, srcIndex)
    setSampleValueAt(dest, destIndex, srcSample)
    srcIndex = srcIndex - 1
```

```
return dest
```

12	25	13
----	----	----

source

12	25	13
----	----	----

dest

Ready for the copy

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
srcIndex = getLength(source)
for destIndex in range(1, getLength(dest) + 1):
    srcSample = getSampleValueAt(source, srcIndex)
    setSampleValueAt(dest, destIndex, srcSample)
    srcIndex = srcIndex - 1
```

```
return dest
```

← You are here

srcIndex

12	25	13
----	----	----

source

destIndex

12	25	13
----	----	----

dest

Do the copy

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
srcIndex = getLength(source)
for destIndex in range(1, getLength(dest) + 1):
    srcSample = getSampleValueAt(source, srcIndex)
    setSampleValueAt(dest, destIndex, srcSample)
    srcIndex = srcIndex - 1
```

```
return dest
```

← You are here

srcIndex

12	25	13
----	----	----

source

destIndex

13	25	13
----	----	----

dest

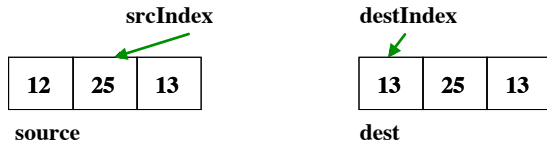
Ready for the next one?

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
    srcIndex = getLength(source)
    for destIndex in range(1, getLength(dest) + 1):
        srcSample = getSampleValueAt(source, srcIndex)
        setSampleValueAt(dest, destIndex, srcSample)
        srcIndex = srcIndex - 1
```

```
    return dest
```

← You are here



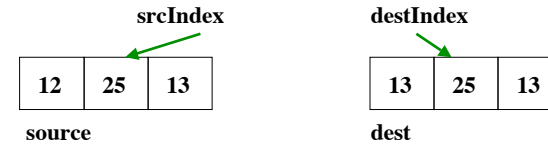
Moving them together

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
    srcIndex = getLength(source)
    for destIndex in range(1, getLength(dest) + 1):
        srcSample = getSampleValueAt(source, srcIndex)
        setSampleValueAt(dest, destIndex, srcSample)
        srcIndex = srcIndex - 1
```

```
    return dest
```

← You are here



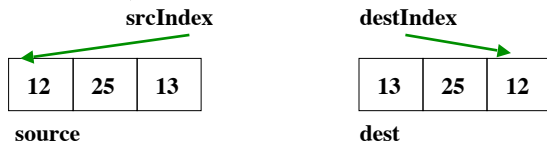
How we end up

```
def playBackward(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
    srcIndex = getLength(source)
    for destIndex in range(1, getLength(dest) + 1):
        srcSample = getSampleValueAt(source, srcIndex)
        setSampleValueAt(dest, destIndex, srcSample)
        srcIndex = srcIndex - 1
```

```
    return dest
```

← You are here



Recipe for halving the frequency of a sound

```
def half(filename):
    source = makeSound(filename)
    dest = makeSound(filename)
```

```
    srcIndex = 1
    for destIndex in range(1, getLength(dest) + 1):
        sample = getSampleValueAt(source, int(srcIndex))
        setSampleValueAt(dest, destIndex, sample)
        srcIndex = srcIndex + 0.5
```

```
    play(dest)
    return dest
```

This is how a sampling synthesizer works!

Here are the pieces that do it

Recipe to double the frequency of a sound

```
def double(filename):
    source = makeSound(filename)
    target = makeSound(filename)
    targetIndex = 1
    for sourceIndex in range(1, getLength(source) + 1, 2):
        value = getSampleValueAt(source, sourceIndex)
        setSampleValueAt(target, targetIndex, value)
        targetIndex = targetIndex + 1
    #Zero out the rest of the target sound -- it's only half full!
    # Zeros are silent.
    for secondHalf in range(getLength(target)/2, getLength(target)):
        setSampleValueAt(target, targetIndex, 0)
        targetIndex = targetIndex + 1
    play(target)
    return target
```

Here's the critical piece:
We skip every other
sample in the source!

What happens if we don't "zero out" the end?

Try this out!

```
def double(filename):
    source = makeSound(filename)
    target = makeSound(filename)
    targetIndex = 1
    for sourceIndex in range(1, getLength(source)+1, 2):
        value = getSampleValueAt(source, sourceIndex)
        setSampleValueAt(target, targetIndex, value)
        targetIndex = targetIndex + 1
    #Clear out the rest of the target sound -- it's only half full!
    #for secondHalf in range(getLength(target)/2, getLength(target)):
    # setSampleValueAt(target, targetIndex, 0)
    # targetIndex = targetIndex + 1
    play(target)
    return target
```

"Switch off" these lines of
code by commenting them out.

Splicing Sounds

- Splicing gets its name from literally cutting and pasting pieces of magnetic tape together
- Doing it digitally is easy (in principle), but painstaking
- Say we want to splice pieces of speech together:
 - Use SoundTool to find the end points of words
 - Copy the samples into the right places to make the words come out as we want them
 - (We can also change the volume of the words as we move them, to increase or decrease emphasis and make it sound more natural.)

Now, it's all about copying

- We have to keep track of the source and target indices, **srcIndex** and **destIndex**

```
destIndex = Where-the-incoming-sound-should-start
for srcIndex in range(startingPoint, endingPoint):
    sampleValue = getSampleValueAt(source, srcIndex)
    setSampleValueAt(dest, destIndex, sampleValue)
    destIndex = destIndex + 1
```

The Whole Splice

```
def splicePreamble():
    file = "/Users/sweat/1315/MediaSources/preamble10.wav"
    source = makeSound(file)
    dest = makeSound(file) # This will be the newly spliced sound
    destSample = 17408 # targetIndex starts after "We the" in the new sound
    for srcSample in range(33414, 40052): # Where the word "United" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt(source, srcSample))
        destSample = destSample + 1
    for srcSample in range(17408, 26726): # Where the word "People" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt(source, srcSample))
        destSample = destSample + 1
    for index in range(1, 1000): #Stick some quiet space after that
        setSampleValueAt(dest, destSample, 0)
        destSample = destSample + 1
    play(dest) #Let's hear and return the result
    return dest
```

What's going on here?

- First, set up a source and target.
- Next, we copy "United" (samples 33414 to 40052) after "We the" (sample 17408)
 - **That means that we end up at $17408 + (40052 - 33414) = 17408 + 6638 = 24046$**
 - **Where does "People" start?**
- Next, we copy "People" (17408 to 26726) immediately afterward.
 - **Do we have to copy "of" to?**
 - **Or is there a pause in there that we can make use of?**
- Finally, we insert a little (1/441-th of a second) of space – 0's

Word	Ending index
We	15730
the	17407
People	26726
of	32131
the	33413
United	40052
States	55510

What if we didn't do that second copy? Or the pause?

```
def splicePreamble():
    file = "/Users/sweat/1315/MediaSources/preamble10.wav"
    source = makeSound(file)
    dest = makeSound(file) # This will be the newly spliced sound
    destSample = 17408 # targetIndex starts after "We the" in the new sound
    for srcSample in range(33414, 40052): # Where the word "United" is in the sound
        setSampleValueAt(dest, destSample, getSampleValueAt(source, srcSample))
        destSample = destSample + 1
    #for srcSample in range(17408, 26726): # Where the word "People" is in the sound
    #setSampleValueAt(dest, destSample, getSampleValueAt(source, srcSample))
    #destSample = destSample + 1
    #for index in range(1, 1000): #Stick some quiet space after that
    #setSampleValueAt(dest, destSample, 0)
    #destSample = destSample + 1
    play(dest) #Let's hear and return the result
    return dest
```

Changing the splice

- What if we wanted to increase or decrease the volume of an inserted word?
 - **Simple! Multiply each sample by something as it's pulled from the source.**
- Could we do something like slowly increase volume (emphasis) or normalize the sound?
 - **Sure! Just like we've done in past programs, but instead of working across *all* samples, we work across only the samples in that sound!**