Audio Effects

By Ralph Quinto and Warren Seto

Background

- We love movies
- Can we do that?
- Create a couple of sound effects

Problem

- Create and implement the following sound effects:
  - Echo
  - Radio
  - Robotic
  - Ghost
  - Darth Vader
- Use Python and DSP libraries

Tools

- Python - High level programming language
  - Portability with all platforms
- SciPy - Open source science computing library
  - Linear Algebra, FFTs, Interpolation, etc
- NumPy - Python library for large computation
  - Multidimensional Arrays, Matrices, etc
- Atom / VS Code

Method

1. Research on what it takes to create an effect
2. Implement and use the DSP libraries to create a effect
3. Compare the group's resulting audio with a similar effect from a movie
4. Debug and tweak
5. Repeat steps 1 to 4 for each type of effect
6. Document

Results

Helper Functions

```
class AudioProcessing(object):

    __slots__ = ('audio_data', 'sample_freq')

    def __init__(self, input_audio_path):
        self.sample_freq, self.audio_data = read(input_audio_path)
        self.audio_data = AudioProcessing.convert_to_mono_audio(self.audio_data)

    def save_to_file(self, output_path):
        '''Writes a WAV file representation of the processed audio data'''
        write(output_path, self.sample_freq, array(self.audio_data, dtype = int16))

    def set_audio_speed(self, speed_factor):
        '''Sets the speed of the audio by a floating-point factor'''
        sound_index = np.round(np.arange(0, len(self.audio_data), speed_factor))
        self.audio_data = self.audio_data[sound_index[sound_index < len(self.audio_data)].astype(int)]

    def set_echo(self, delay):
        '''Applies an echo that is 0...<input audio duration in seconds> seconds from the beginning'''
        output_audio = np.zeros(len(self.audio_data))
        output_delay = delay * self.sample_freq

        for count, e in enumerate(self.audio_data):
            output_audio[count] = e + self.audio_data[count - int(output_delay)]

        self.audio_data = output_audio
```

```
    def set_volume(self, level):
        '''Sets the overall volume of the data via floating-point
        output_audio = np.zeros(len(self.audio_data))

        for count, e in enumerate(self.audio_data):
            output_audio[count] = (e * level)

        self.audio_data = output_audio

    def set_reverse(self):
        '''Reverses the audio'''
        self.audio_data = self.audio_data[::-1]

    def set_audio_pitch(self, n, window_size=2**13, h=2**11):
        '''Sets the pitch of the audio to a certain threshold'''
        factor = 2 ** (1.0 * n / 12.0)
        self._set_stretch(1.0 / factor, window_size, h)
        self.audio_data = self.audio_data[window_size:]
        self.set_audio_speed(factor)
```

```
def set_lowpass(self, cutoff_low, order=5):
    '''Applies a low pass filter'''
    nyquist = self.sample_freq / 2.0
    cutoff = cutoff_low / nyquist
    x, y = signal.butter(order, cutoff, btype='lowpass', analog=False)
    self.audio_data = signal.filtfilt(x, y, self.audio_data)

def set_highpass(self, cutoff_high, order=5):
    '''Applies a high pass filter'''
    nyquist = self.sample_freq / 2.0
    cutoff = cutoff_high / nyquist
    x, y = signal.butter(order, cutoff, btype='highpass', analog=False)
    self.audio_data = signal.filtfilt(x, y, self.audio_data)

def set_bandpass(self, cutoff_low, cutoff_high, order=5):
    '''Applies a band pass filter'''
    cutoff = np.zeros(2)
    nyquist = self.sample_freq / 2.0
    cutoff[0] = cutoff_low / nyquist
    cutoff[1] = cutoff_high / nyquist
    x, y = signal.butter(order, cutoff, btype='bandpass', analog=False)
    self.audio_data = signal.filtfilt(x, y, self.audio_data)
```

How we built the effects?

Echo

```
@staticmethod
def echo(input_path, output_path):
    '''Applies an echo effect to a given
    sound = AudioProcessing(input_path)
    sound.set_echo(0.09)
    sound.save_to_file(output_path)
```

Radio

```
@staticmethod
def radio(input_path, output_path):
    '''Applies a radio effect to a give
    sound = AudioProcessing(input_path)
    sound.set_highpass(2000)
    sound.set_volume(4)
    sound.set_bandpass(50, 2600)
    sound.set_volume(2)
    sound.save_to_file(output_path)
```

How we built the effects? (continued)

Ghost

```
@staticmethod
def ghost(input_path, output_path):
    '''Applies a ghostly halloween effec
    sound = AudioProcessing(input_path)
    sound.set_reverse()
    sound.set_echo(0.05)
    sound.set_reverse()
    sound.set_audio_speed(.70)
    sound.set_audio_pitch(2)
    sound.set_volume(8.0)
    sound.set_bandpass(50, 3200)
    sound.save_to_file(output_path)
```

Robotic

```
@staticmethod
def robotic(input_path, output_path):
    '''Applies a robotic effect to a giv
    sound = AudioProcessing(input_path)
    sound.set_volume(1.5)
    sound.set_echo(0.01)
    sound.set_bandpass(300, 4000)
    sound.save_to_file(output_path)
```

How we built the effects? (continued)

Darth Vader

```
@staticmethod
def darth_vader(input_path, output_path):
    '''Applies a Darth Vader effect to a g
    sound = AudioProcessing(input_path)
    sound.set_audio_speed(.8)
    sound.set_echo(0.02)
    sound.set_lowpass(2500)
    sound.save_to_file(output_path)
```

Demo

Summary

- Python + SciPy + NumPy
- Produces cinema audio effects
  - Echo, Radio, Robotic, Ghost and Darth Vader
- A practical application of DSP
  - Filters, echos, etc
- Create an API called: *AudioProcessing* / *AudioEffect*

Audio Effects

https://github.com/nextseto/PythonAudioEffects

By Ralph Quinto and Warren Seto