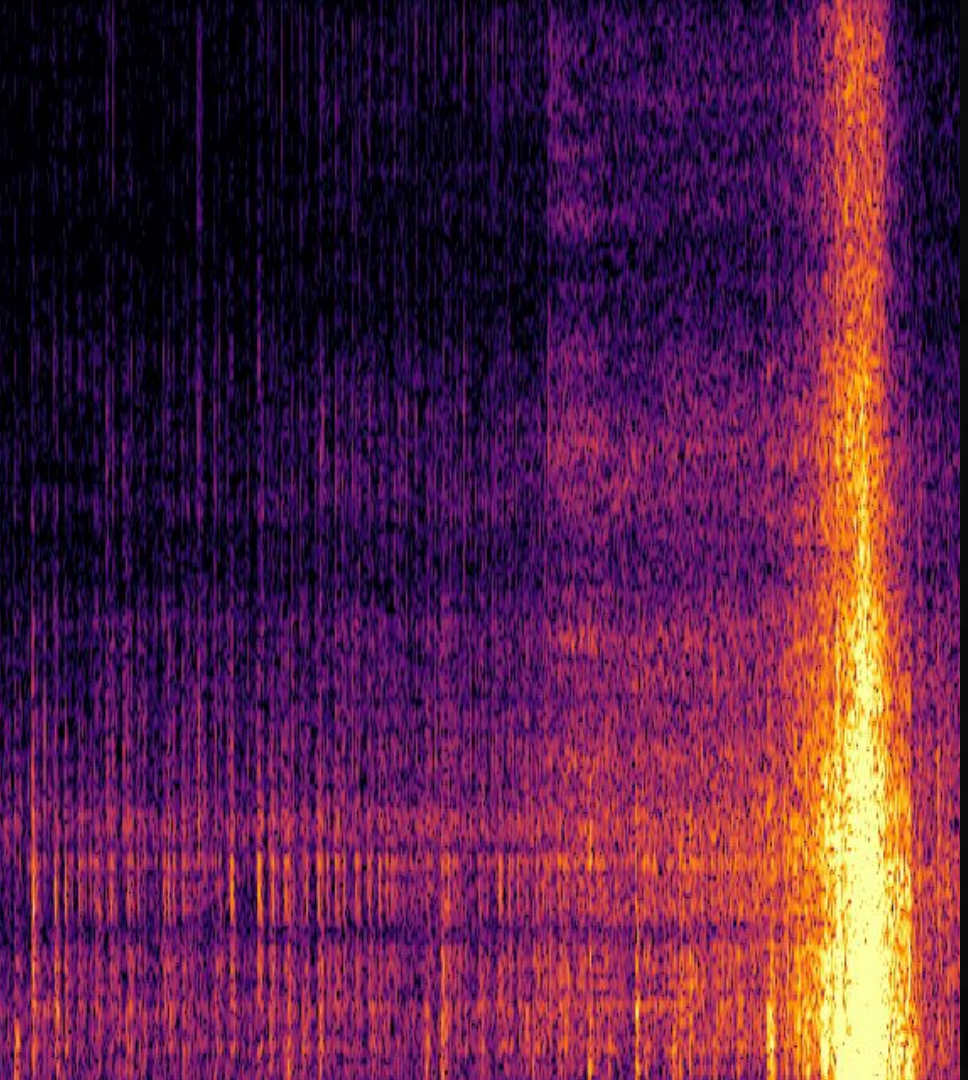
sonification for PLANETS

Jonathan Deng, Vaidehi Karve, Jeffrey Lin



Sonification

Our project was with PLANETS, in collaboration with NASA, USGS, NAU CSTL, Museum of Science Boston, EiE, and WestEd.

Sonification is “the use of non-speech audio to convey information or perceptualize data.” Essentially, it is the process of converting different waves and signals to audio.

We converted the spectra of minerals and leaves to sound. Other examples include sonifying galaxies, the Aurora Borealis, EM waves, and more!

PLANETS

NASA

PLANETS

NAU NORTHERN ARIZONA UNIVERSITY
Center for Science Teaching and Learning

USGS science for a changing world
USGS Astrogeology Science Center

EiE Engineering is Elementary

WestEd
WestEd

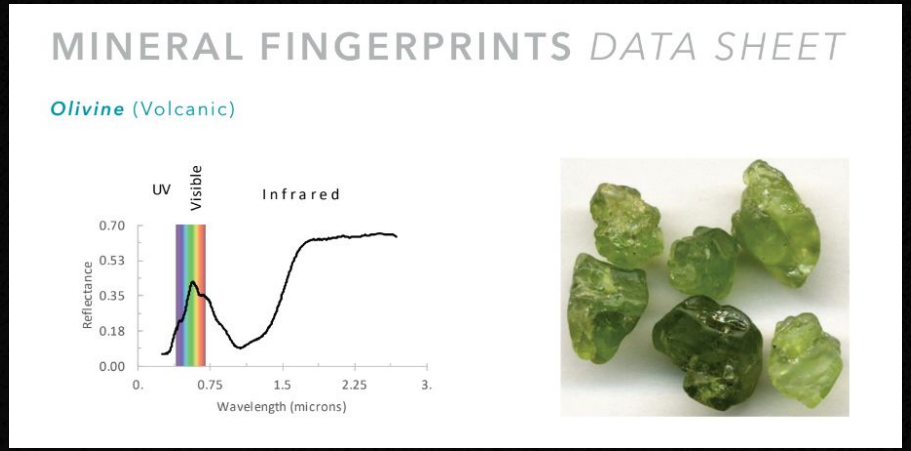
BACKGROUND

PLANETS (Planetary Learning Advancing the Nexus of Engineering, Technology, and Science) is an out-of-school-time program for students from grades 3-8. PLANETS is a cooperative agreement between NASA and PLANETS partners, including USGS (United States Geological Survey), NAU Center for Science Teaching and Learning (CSTL), Museum of Science Boston EiE, and WestEd. They focus on STEM education—specifically NASA's planetary science and engineering. With support from our professors and PLANETS, we are working to help incorporate inclusivity into their lessons.

Their Remote Sensing Unit simulates life on Mars; students investigate the surface of the planet and search for rover landing sites. This includes analyzing data of six different minerals—olivine, pyroxene, kaolinite, nontronite, kieserite, and magnesite—using their spectra. This activity is entirely visual-based, barring those with visual disabilities from participating. Using Pure Data and Python, we have developed four ways to make the lesson more accessible to blind and low-vision youth. We sonified the existing spectral graphs of the six minerals, turning them into audio files that are available for downloading to educators.

MATERIALS

- Google Colab
- USGS Spectral Library
- Python Librosa, NumPy, SciPy, IPython, Matplotlib
- USGS Mineral Fingerprints Data Sheet
- Pure Data
- Fast Fourier Transform (FFT) algorithm



PROCEDURE

Python Pre-Processing

- Load in the data downloaded from the USGS database
- Perform pre-processing on the data, including changing scientific notation to floats, converting frequency to wavelength, changing negatives to 0
- Save the data as new files in folder for later use

```
[ ] import numpy as np
```

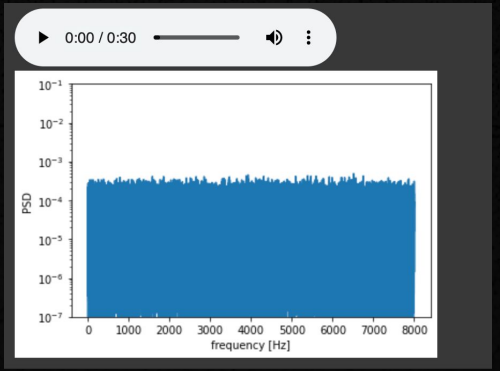
```
[ ] def load_file(filename):
      data = np.loadtxt(filename, skiprows=1)
      data[np.where(data < 0)] = 0
      return np.flipud(data)

    pretty_print = lambda x: np.format_float_positional(x, trim="-")
    def save_file(data, filename):
      with open(filename, 'w') as f:
        for d in data:
          f.write(pretty_print(d) + "\n")
```

```
[ ] filenames = ['Kaolinite.txt',
                 'Kaolinite_1.5.txt',
                 'Kieserite.txt',
                 'Kieserite_1.5.txt',
                 'Magnesite.txt',
                 'Magnesite_1.5.txt',
                 'Nontronite.txt',
                 'Nontronite_1.5.txt',
                 'Olivine.txt',
                 'Olivine_1.5.txt',
                 'Pyroxene.txt',
                 'Pyroxene_1.5.txt']

    for filename in filenames:
      data = load_file('data2/' + filename)
      parts = filename.split('.')
      out_filename = 'data2/' + '.'.join(parts[:-1]) + '_clean.' + parts[-1]
      print(out_filename)
      save_file(data, out_filename)
```

Python Filters

- Load in the cleaned-up files and use the fast fourier transform to manipulate individual frequencies
- Run a periodogram to generate filters based on mineral spectra
  - Sampling frequencies
  - FIR (finite impulse response) filter orders
  - Duration

```
IPython.display.display(ipd.Audio(data=x, rate=fs))

f, Pxx_den = signal.periodogram(x, fs)
plt.semilogy(f, Pxx_den)
plt.ylim([1e-7, 1e-1])
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
```




```
[ ] import numpy as np
    from scipy import signal
    import matplotlib.pyplot as plt
    import IPython.display as ipd
    import IPython
    import librosa
```

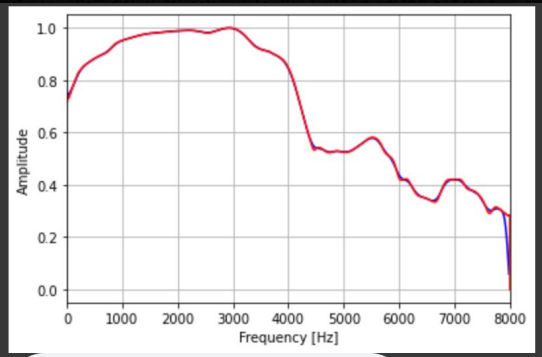
```
[ ] def load_file(filename, fs):
      data = np.loadtxt(filename, skiprows=1)
      data = np.insert(data, 0, 0)
      data = data[np.where(data >= 0)]
      m = len(data)
      data = np.sqrt(np.flipud(data))
      return np.linspace(0, fs/2, m), data/np.max(data)

    def design_filter(ff, fw, fs, n):
      # design filter
      b = signal.firwin2(n, ff, fw, fs=fs)
      return b, 1

    def white_noise(T, fs, seed=None):
      N = int(T * fs)
      rng = np.random.default_rng(seed=seed)
      return rng.uniform(-1, 1, N)
      # return rng.standard_normal(N)
```

```
[ ] fs = 16000  # sampling frequency (Hz)
    n = 200     # FIR filter order
    T = 30      # duration of signals (s)
```

```
[ ] # generate white noise
    x = white_noise(T, fs)
```

Designing Filters

```
filenames = ['Kaolinite_clean.txt',
             'Kaolinite_1.5_clean.txt',
             'Kieserite_clean.txt',
             'Kieserite_1.5_clean.txt',
             'Magnesite_clean.txt',
             'Magnesite_1.5_clean.txt',
             'Nontronite_clean.txt',
             'Nontronite_1.5_clean.txt',
             'Olivine_clean.txt',
             'Olivine_1.5_clean.txt',
             'Pyroxene_clean.txt',
             'Pyroxene_1.5_clean.txt']

filename = filenames[2]

# load file
ff, fw = load_file('data2/' + filename, fs)

# design filter
b, a = design_filter(ff, fw, fs, n)

# compare frequency responses
w, h = signal.freqz(b, a, fs=fs)
plt.plot(w, abs(h), 'b', ff, fw, 'r')
plt.xlim(0, fs/2)
plt.grid()
plt.ylabel('Amplitude')
plt.xlabel('Frequency [Hz]');
plt.show()

# filter noise
y = signal.lfilter(b, a, x)

IPython.display.display(ipd.Audio(data=y, rate=fs))

f, Pxx_den = signal.periodogram(y, fs)
plt.plot(f, Pxx_den, ff, .5*1e-3*fw)
plt.xlim(0, fs/2)
plt.grid()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
```

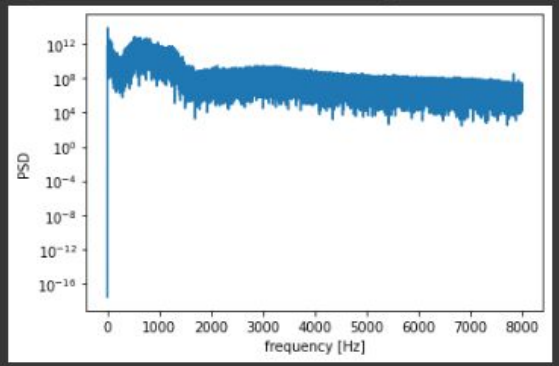


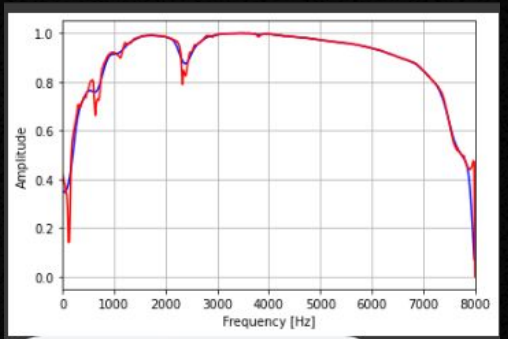
- Array of files used to design filters based on the previous function using filter frequencies, filter weights, fourier series
- A periodogram approximates the spectral data from the minerals, which is used on the white noise as a filter using the fast fourier transform

Generated Audio

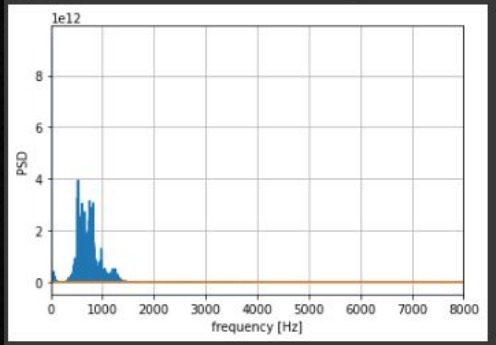
```
IPython.display.display(ipd.Audio(data=x, rate=fs))

f, Pxx_den = signal.periodogram(x, fs)
plt.semilogy(f, Pxx_den)
# plt.ylim([1e-7, 1e-1])
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
```

0:00 / 0:30

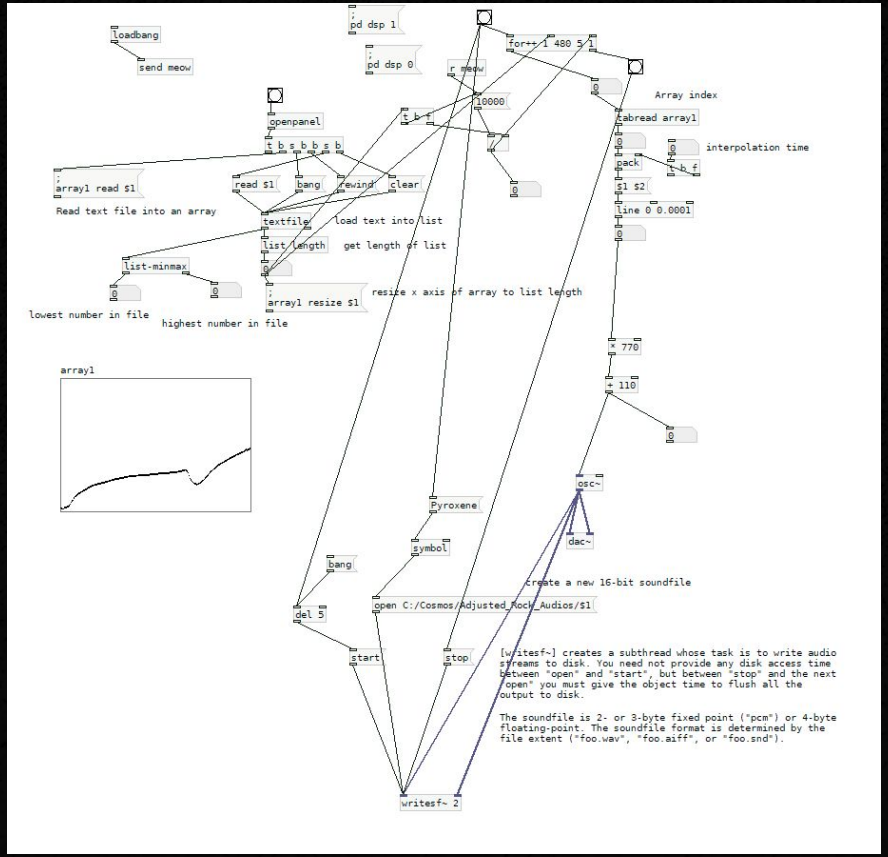




0:00 / 0:30



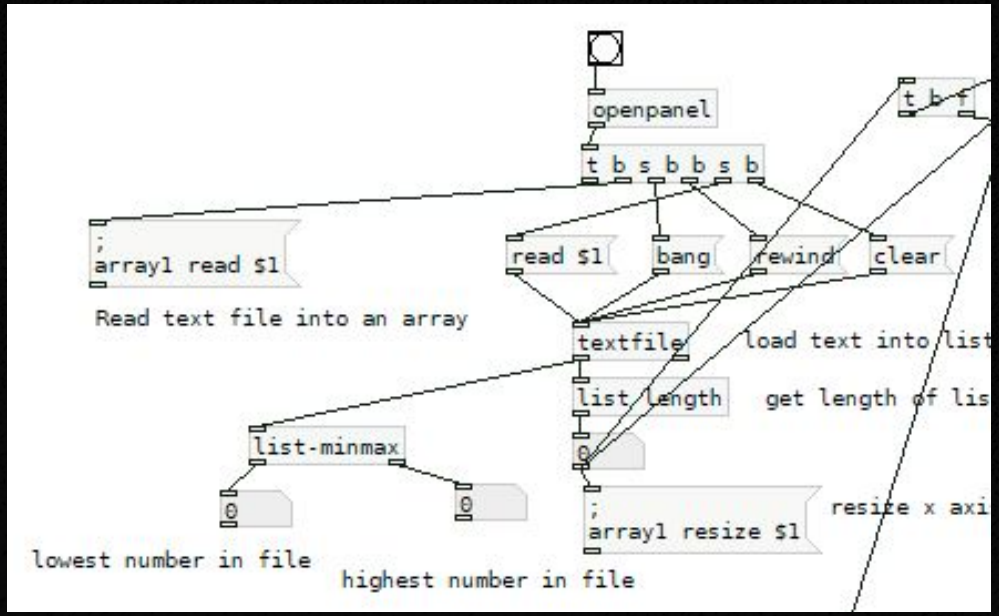
Pure Data


loadbang
send meow
pd dsp 1
pd dsp 0
r meow
for++ 1 480 5 1
10000
Array index
tabread array1
openpanel
t b s b b s b
read $1
bang
rewind
clear
array1 read $1
Read text file into an array
textfile
load text into list
list length
get length of list
list-minmax
array1 resize $1
resize x axis of array to list length
lowest number in file
highest number in file
interpolation time
pack
$1 $2
line 0 0.0001
* 770
+ 110
array1
osc~
dac~
Pyroxene
symbol
bang
create a new 16-bit soundfile
open C:/Cosmos/Adjusted_Rock_Audios/$1
del 5
start
stop
writesf~ 2
[writesf~] creates a subthread whose task is to write audio streams to disk. You need not provide any disk access time between "open" and "start", but between "stop" and the next "open" you must give the object time to flush all the output to disk.
The soundfile is 2- or 3-byte fixed point ("pcm") or 4-byte floating-point. The soundfile format is determined by the file extent ("foo.wav", "foo.aiff", or "foo.snd").

Loading the Array

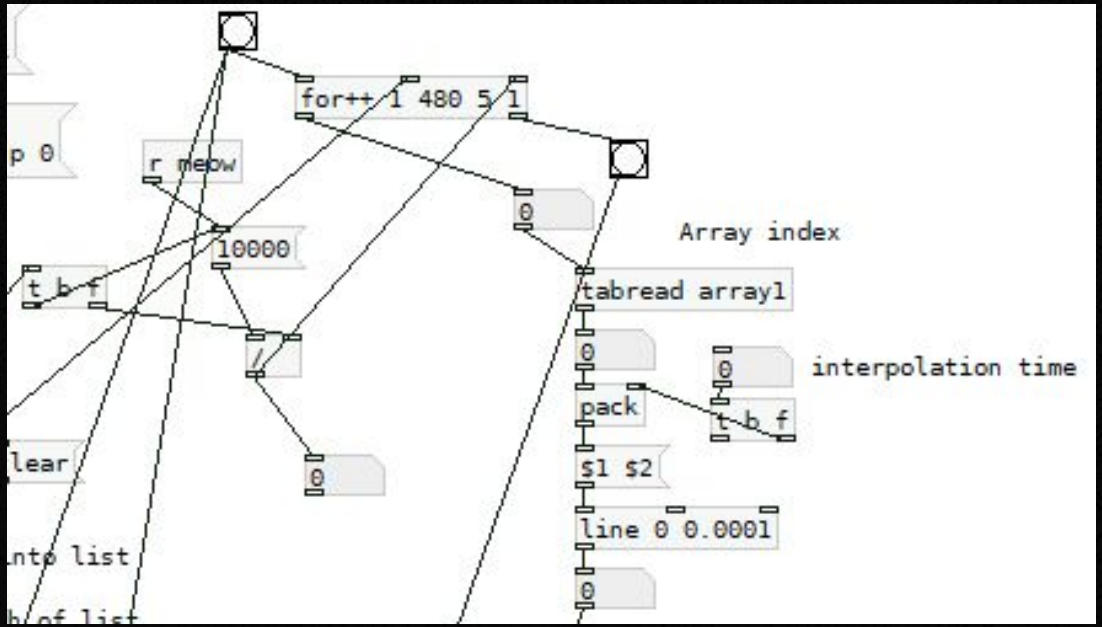
Opens up the data as a txt file using openpanel, and loads it into an array.

Parameters such as minimum and maximum values and list length will be used later.



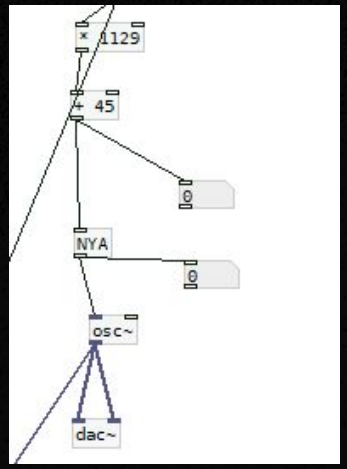
Reading the Array

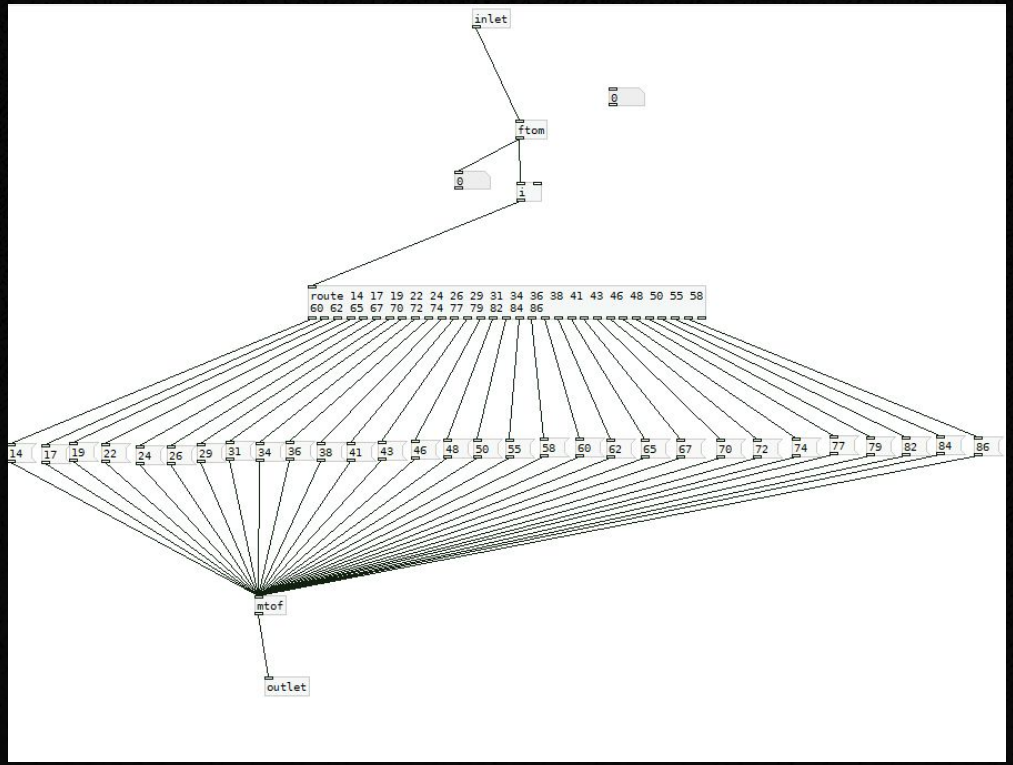
For++ object iterates through the array using the tabread object. The length of the loop is the length of the array, and the time between each incrementation is determined by the list length to keep the time at 10 seconds.



oscillator

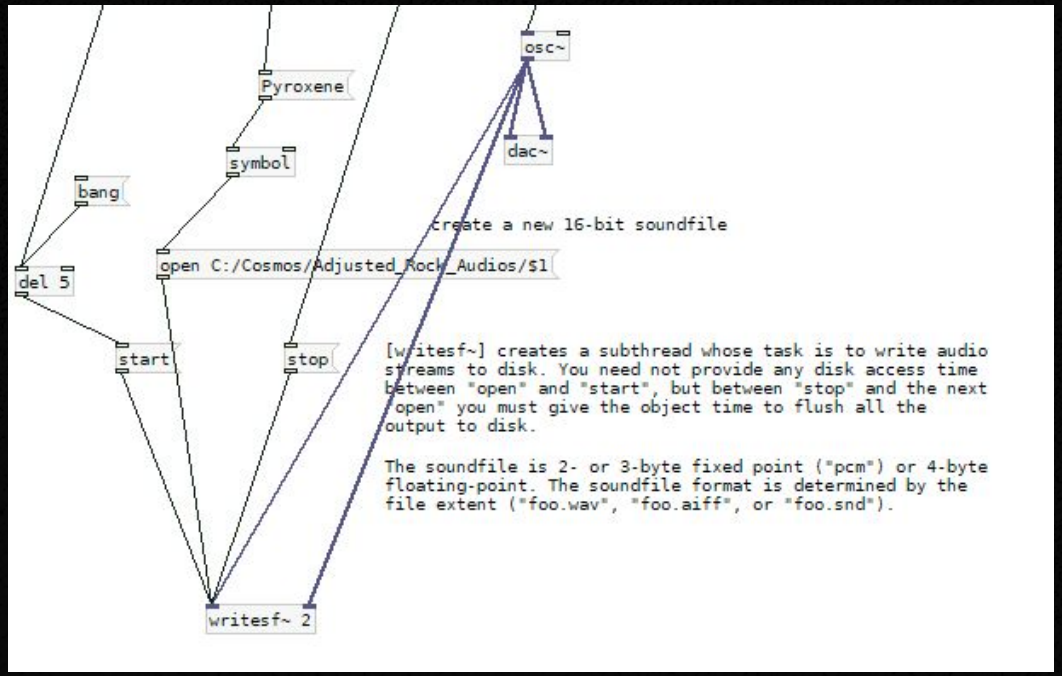
This approach is the same as the previous one, but adds a pentatonic scale.
Multiplies the reflectance at each wavelength to a frequency that is audible. This can now be run through the midi filter if needed, and then played on an oscillator.





Writing to Disk

Use the bang trigger to initiate the open object and create a new .wav file, and start writing into the file. Once the for++ object is complete, the bang triggers a stop message that stops writing to disk.



After COSMOS

We liked the Pure Data options, but found that the Python approaches were extremely difficult to make sense of. Considering that our intended audience consists of elementary and middle schoolers, it is important that the output produced is easy to understand and distinguish. The students must be able to identify and match the frequencies to their respective materials, and this is best done through Pure Data.

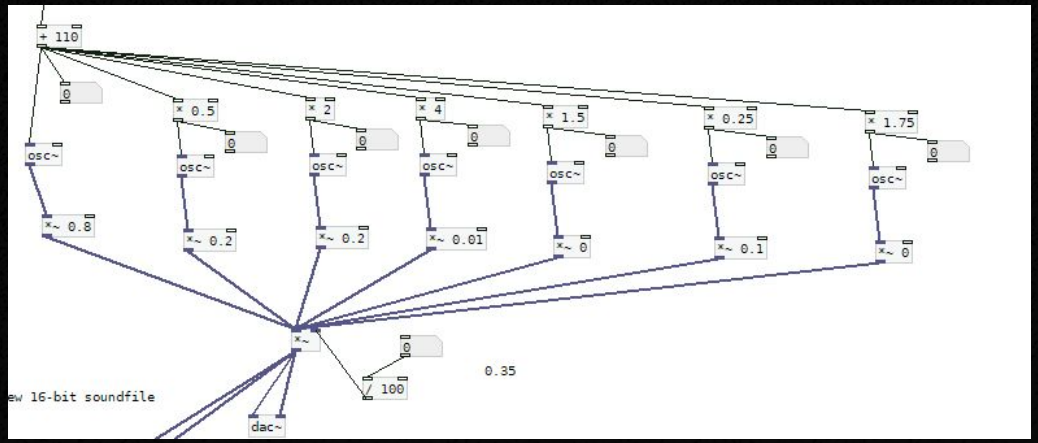
Since the end of COSMOS, we have sonified the spectra of three maple leaves. We also added harmonics using Pure Data for both the Mars minerals and the leaves.

We have refined the Python code and the audio filters to produce different, euphonic outputs. Additionally, we plan to incorporate stereo sounds and a series of different instruments to represent the spectra.

After Cosmos

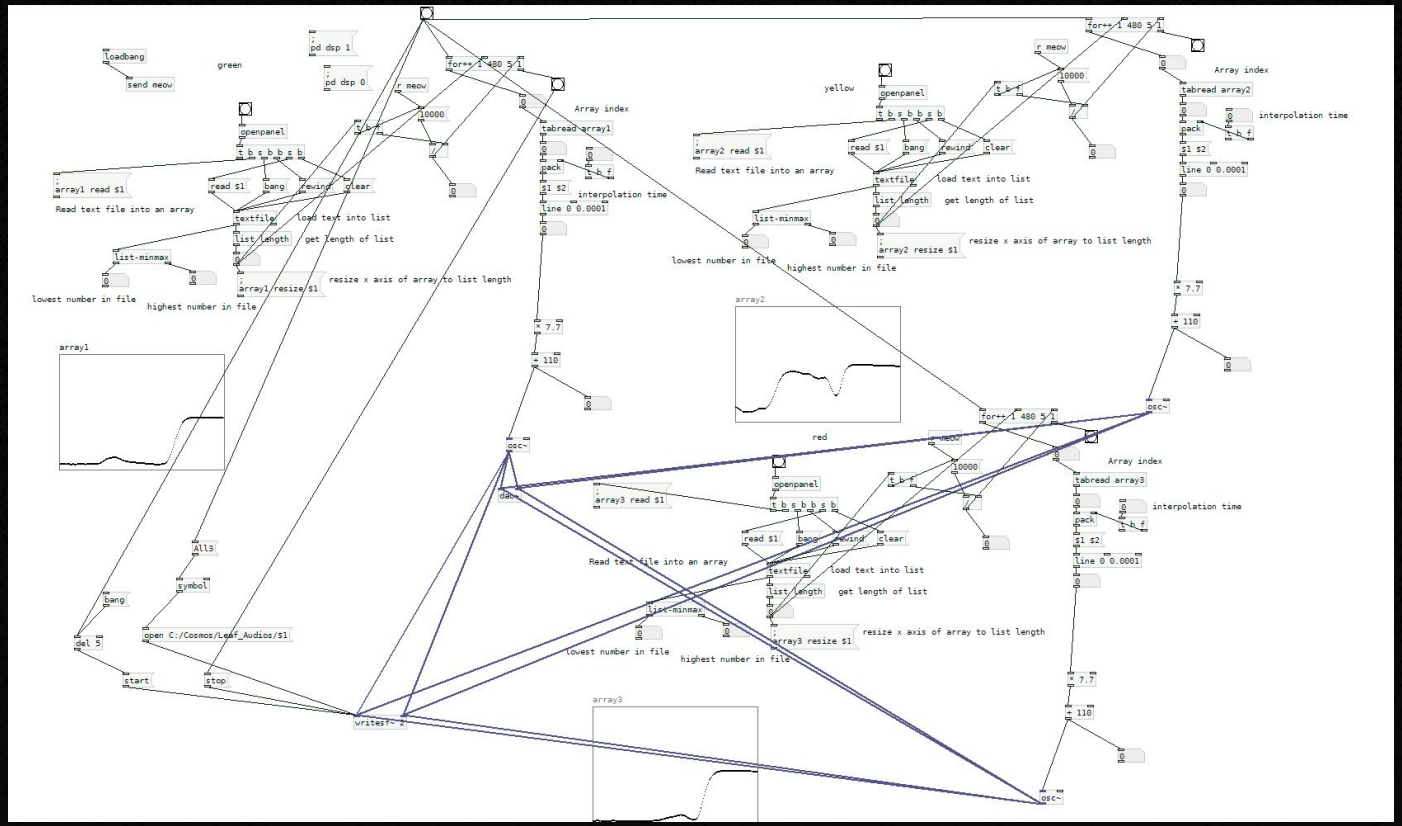
We adjusted the frequencies from 45 hz - 1174 hz to 110 hz - 880 hz to greatly reduce the range of the audio and prevent it from being too high pitched or low pitched. This also makes it more accessible on cheaper speakers, which is what they’re likely to be used on. Multiple errors were fixed such as incorrect scaling of frequency made it difficult to compare between different spectra, and the audio was re-recorded in dual channel.

Harmonics



Split the signal into multiple different frequencies separated by octaves, which makes the sound more pleasing. Our testing found that this timbre of sound was the least distracting.

ALL 3 Leaves


loadbang
send meow
green
pd dsp 1
pd dsp 0
r meow
for++ 1 480 5 1
Array index
tabread array1
openpanel
t b s b b s b
read $1
bang
rewind
clear
array1 read $1
Read text file into an array
textfile
load text into list
list length
get length of list
list-minmax
lowest number in file
highest number in file
array1 resize $1
resize x axis of array to list length
pack
$1 $2
interpolation time
line 0 0.0001
* 7.7
+ 110
array1
osc~
dac~
All3
symbol
bang
del 5
open C:/Cosmos/Leaf_Audios/$1
start
stop
writesf~ 2
yellow
tabread array2
array2 read $1
array2 resize $1
array2
red
tabread array3
array3 read $1
array3 resize $1
array3

What we learned

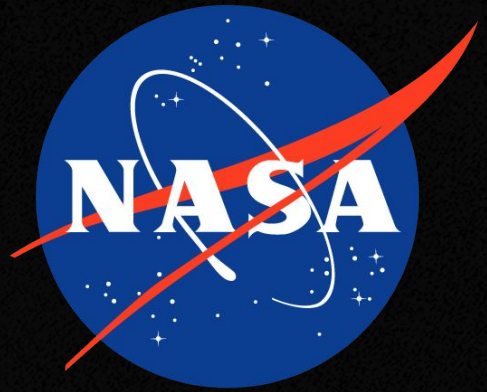
- We learned the importance of making lessons accessible to everyone
- We learned about new tools like Pure Data, SciPy, and Librosa

Acknowledgements

Thank you so much to everyone who helped us along the way!

- Lori Pigue
- Lori Ann Rubino-Hare
- Victor Minces
- NASA, USGS, NAU CSTL, Museum of Science Boston EiE, and WestEd
- Maurício de Oliveira
- Joe Cantrell
- Gualter Moura
- Tornike Karchkhadze
- Benjamin Redlawsk
- Uday Mehra

Thank you!



PLANETS

NAU NORTHERN ARIZONA UNIVERSITY
Center for Science Teaching and Learning

USGS science for a changing world
USGS Astrogeology Science Center

EiE Engineering is Elementary

WestEd
WestEd