INTRODUCTION TO AUDIO STEGANOGRAPHY

Chris Lam, Jason Liu

Audio Files

Waveform Audio File (WAV)

- Stores raw audio samples
- Usually uncompressed
- Easily able to be converted to other forms but also bigger file sizes

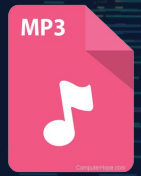
Structure:

44 Bytes Header

Audio Data

MPEG Audio Layer 3 (MP3)

- Highly compressed audio by using lossy compression (trades worse audio quality for smaller file sizes)
- Lossy compression gets rid of audio that is inaudible to the human ear



MPEG-4 Part 14 (MP4)

- Multimedia container - can hold audio, video, images, and other forms of data

Glossary

Sampling Rate - Samples per one second (Hz)

Sample - Amplitude of waveform at a given time (Higher amplitude = louder sounds)

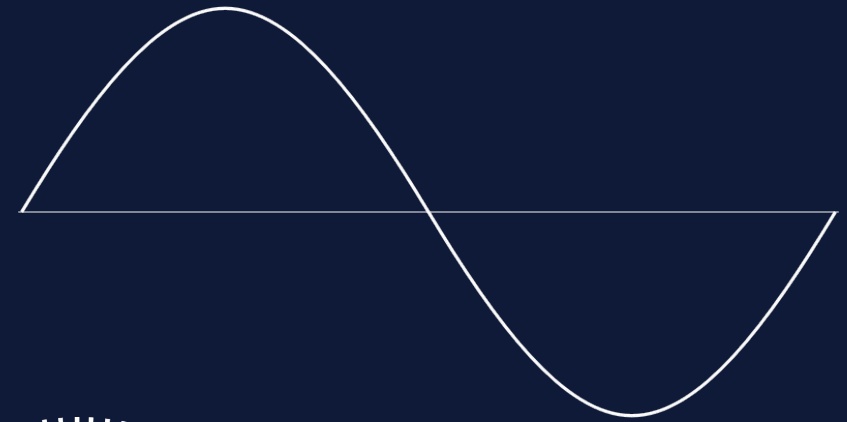
Channels - Audio steam

- Mono - 1 channel
- Stereo - 2 channel

Frame - Set of samples at a given time

Nyquist–Shannon sampling theorem - smart person said that to accurately recreate a WAV, sampling rate should be at least twice the highest frequency

Old white dude with glasses so it's probably Nyquist idk I didn't double check



Generate Wav

In NUMPY we trust

$y = A \sin(\omega t + \varphi)$

A: amplitude

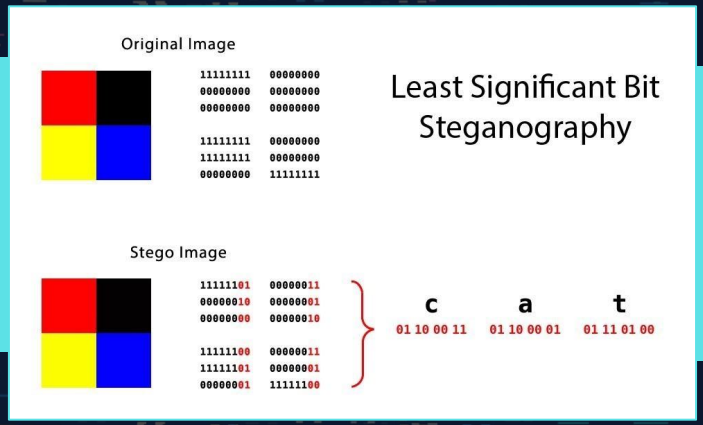
ω: angular frequency (2π* frequency)

t: time

φ: phase

```
samples = (amplitude * np.sin(2 * np.pi * freq * t))
```

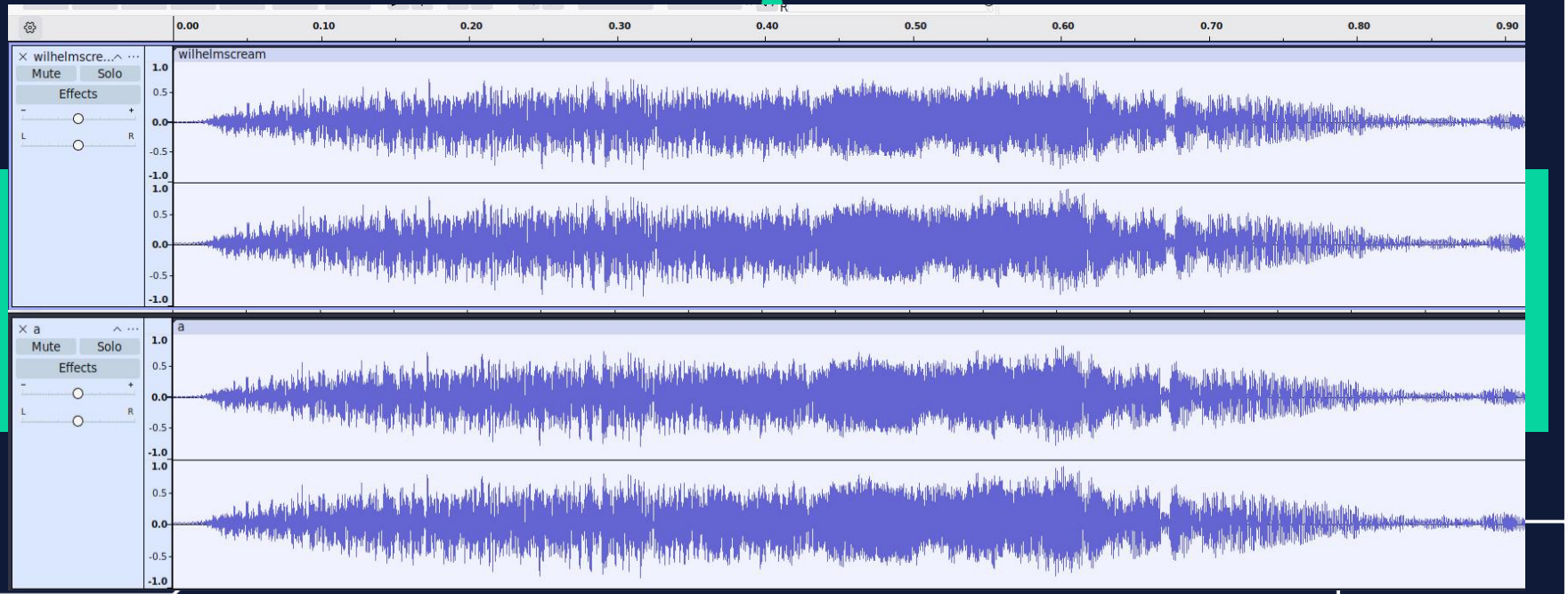
Least Significant Bit (LSB)



If you recall from visual steganography, we changed the color of some pixels in the image by encoding some bits into their least significant value.

The bytes in a WAV file are the samples of the audio file, meaning, if we encode with LSB we will change the loudness of certain parts of the audio. WAV files are little endian, meaning we will have to change the first bit.

Comparison


0.00
0.10
0.20
0.30
0.40
0.50
0.60
0.70
0.80
0.90
wilhelmscre...
Mute
Solo
Effects
wilhelmscream
a
1.0
0.5
0.0
-0.5
-1.0

Echo Hiding

- Literal witchcraft
- I hate it so much it made me bilingual: “我讨厌 echo hiding”
  - Translation: “I hate echo hiding”
- Take segments of the original audio, delay the segments using one of two predefined delays, and then add the segments back into the original audio
  - The distinct time intervals correspond to what bits will be turned on when decoding

What if I don’t have the original audio?

- **You cry**
- **Autocorrelation**

Echo Hiding Pt 2

Phase changes are harder for human ears to perceive

- Remember Ohm? Ohm's Acoustic Law (we analyze sounds through frequency)
- Temporal masking (loud sound masks quiet sound)

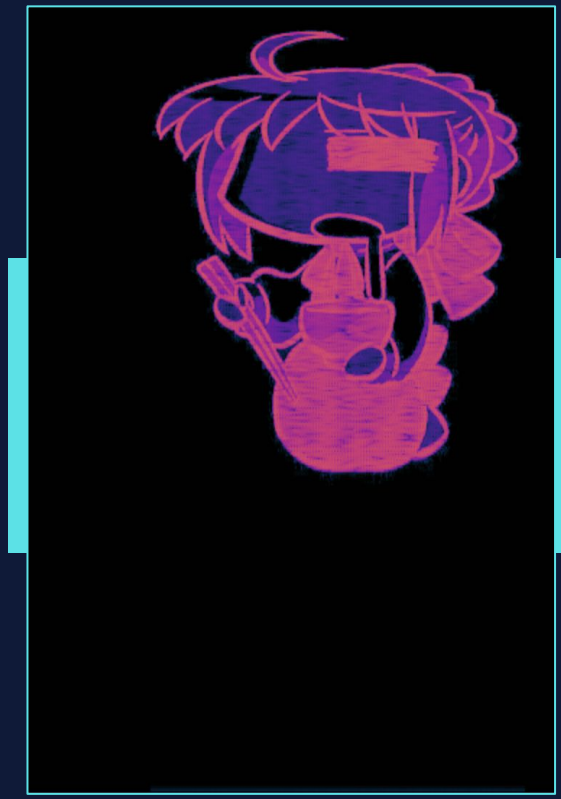
Decode:

Loop through the segments

Dot product of two signals shows how similar the two signals are to one another

Compare dot products between corresponding delay times

Image-Audio Conversion



Images can be converted into an audio file and be hidden in the spectrogram

What is a spectrogram?

- A Spectrogram consists of a set of 2D arrays
  - Time Buckets: X-Axis
  - Frequency Buckets: Y-Axis
  - Magnitude: Variable 1
  - Phase: Variable 2

01. Image Preparation

- Convert Image into Array of Pixels.
- Create Array of Normalized Values Grayscale for the Pixels
- Flip Image as STFT and Image treat 2D arrays differently

```
# Load the image and accquires its size
image = Image.open(imageName)
image = image.convert("RGB")
width, height = image.size

# Convert to grayscale
gray_image = Image.new("RGB", (width, height))
pixels = image.load()
gray_pixels = gray_image.load()

for i in range(height):
    for j in range(width):
        r, g, b = pixels[j, i]
        gray = int(0.299 * r + 0.587 * g + 0.114 * b)
        gray_pixels[j, i] = (gray, gray, gray)

# Create normalized magnitude array
normalized_magnitudes = np.zeros((height, width))

for i in range(height):
    for j in range(width):
        r, _, _ = gray_pixels[j, i]
        normalized_magnitudes[i][j] = r / 255.0

# Flip the image vertically to match spectrogram orientation
normalized_magnitudes = np.flipud(normalized_magnitudes)
```

02. Audio Preparation

- Load an audio file
- Perform Short-Time Fourier Transformation to get Spectrogram Information
- Acquire Array of Magnitude and Phase values

```
# Creating base audio
y, sr = librosa.load(audioName, sr=None)

# Convert audio to spectrogram (Short-Time Fourier Transf
n_fft = (int)(2048 * frequency_resolution)      # default
hop_length = (int)(512 / (time_resolution*1.0)) # smaller
D = librosa.stft(y, n_fft=n_fft, hop_length=hop_length)
magnitude, phase = np.abs(D), np.angle(D)
```

03. Creating Magnitudes

- Resize Image to be size of STFT Magnitude Array
- Optionally change mode so that white or black is the main color
- Change the strength of the values for visibility
- Scale values to maximum magnitude

```
# Resize the image magnitudes to match the spectrogram shape
image_resized = np.array(
    Image.fromarray(
        (normalized_magnitudes * 255).astype(np.uint8)
    ).resize(
        magnitude.shape[::-1]  # PIL expects (width, height)
    )
)
image_resized = image_resized.astype(np.float32) / 255.0  # Normalize

# Optionally invert image if you want darker = louder
if(mode == "Dark"):
    image_resized = 1.0 - image_resized

# Weaken the brightness of the image
image_resized = image_resized.astype(np.float32) / (100.0/visibility)

# Scale image magnitudes to match the energy range of the original spe
image_scaled = image_resized * np.max(magnitude)

# Combine image magnitudes with original phase
D_modified = image_scaled * np.exp(1j * phase)
```

04. Form into Audio

- Combine new Magnitude with original phase
- Use Inverse Short-Time Fourier Transformation

Save Audio

```
# Combine image magnitudes with original phase
D_modified = image_scaled * np.exp(1j * phase)

# Convert back to time domain
y_modified = librosa.istft(D_modified)

# Save the modified audio
import soundfile as sf
sf.write(outputName, y_modified, sr)
```

Image-Audio Embedding

Images can also be embedded within the spectrogram of an audio file.

Well Known Example:

Doom Soundtrack

03. Applying Magnitudes

- Add normalized values to the already existing audio array
- Modifications done to visibility

```
# Quick Check if image will fit
end_row = start_row + target_height
end_col = start_col + target_width
assert end_row <= magnitude.shape[0] and end_col <= magnitude.shape[1], "Image doesn't fit!"

# adding to magnitude
magnitude[start_row:end_row, start_col:end_col] += image_resized * np.max(magnitude) / (100.0/visibility)
```

Strength

- Can be very subtle when applied correctly
- Not viewable on first glance as spectrogram has to be contorted

Weakness

- Very Documented Method leading to it being easier to detect
- Image Quality and Visibility Degradation with high quality resulting in louder disturbances

How to Use

Color Priority Mode (Dark, Light)

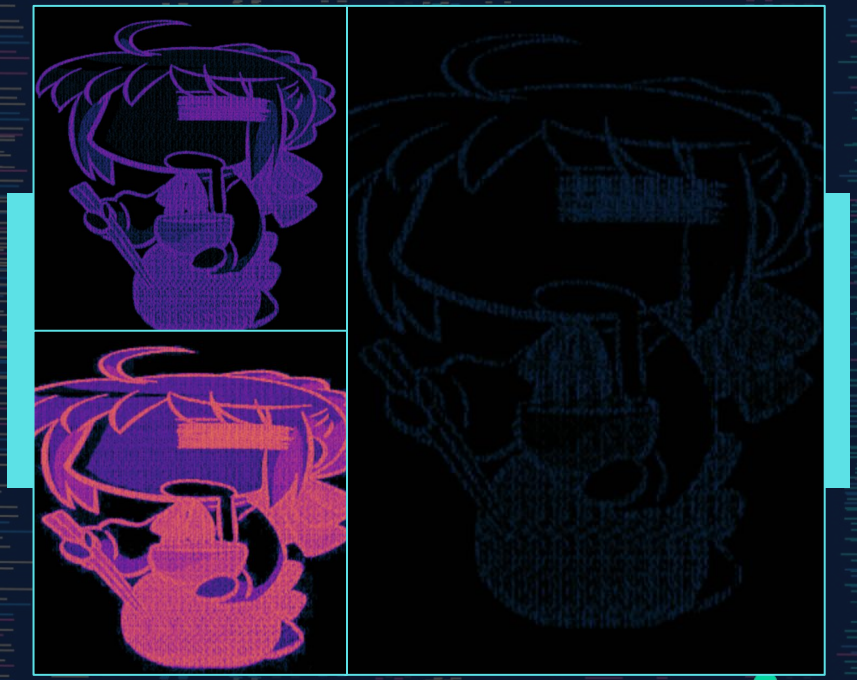
- Whether the image should be drawn based on dark sections or light sections of images

Visibility Percentage

- How bright should the entire image be on the spectrogram

Time Resolution Multiplier & Frequency Resolution Multiplier

- Multipliers on Image Quality
- Increases Run Time



How to Use

Color Priority Mode (Dark, Light)

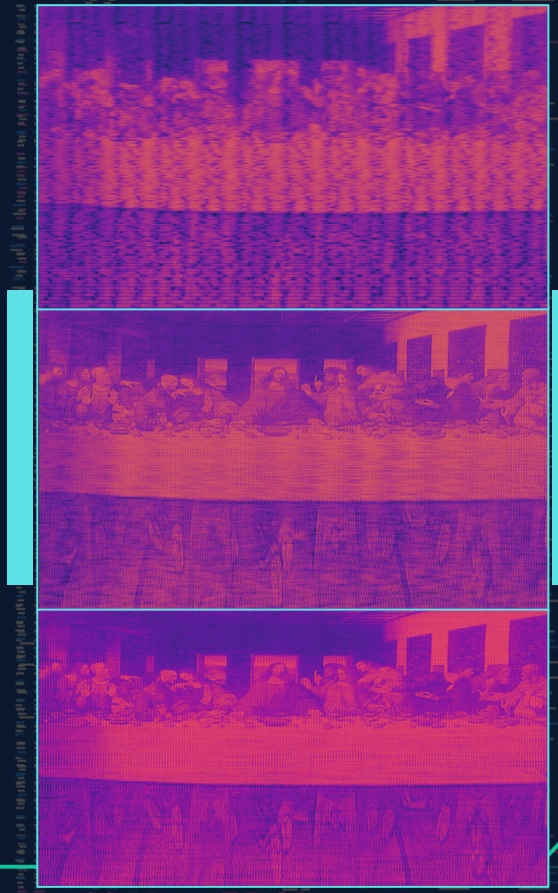
- Whether the image should be drawn based on dark sections or light sections of images

Visibility Percentage

- How bright should the entire image be on the spectrogram

Time Resolution Multiplier & Frequency Resolution Multiplier

- Multipliers on Image Quality
- Increases Run Time



How to Use

Color Priority Mode (Dark, Light)

- Whether the image should be drawn based on dark sections or light sections of images

Visibility Percentage

- How bright should the entire image be on the spectrogram

Time Resolution Multiplier & Frequency Resolution Multiplier

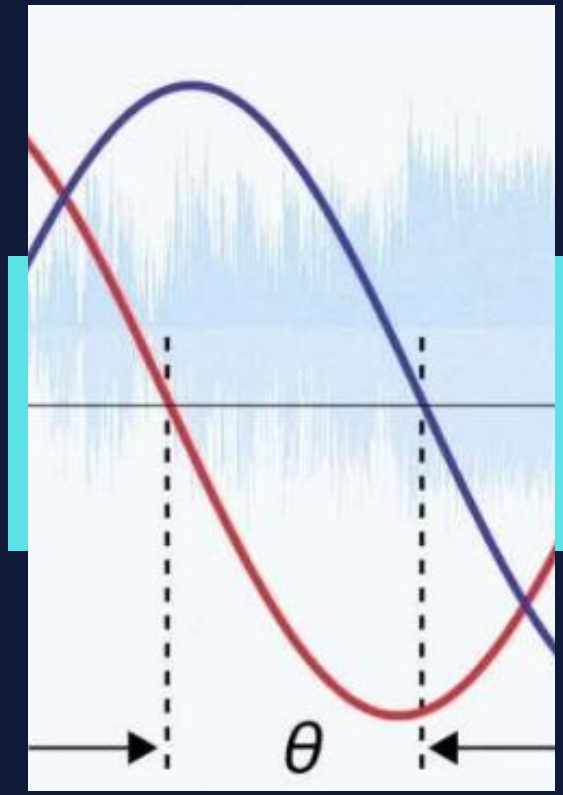
- Multipliers on Image Quality
- Increases Run Time

Phase Encoding

Bytes can be encrypted with an audio file by modifying the phases found to indicate 0s and 1s

How is this done?

- Phases are modified into a new audio file
- New audio file is compared to old to find a difference
- Phase changes are hard for the human ear to perceive



THE END

yay