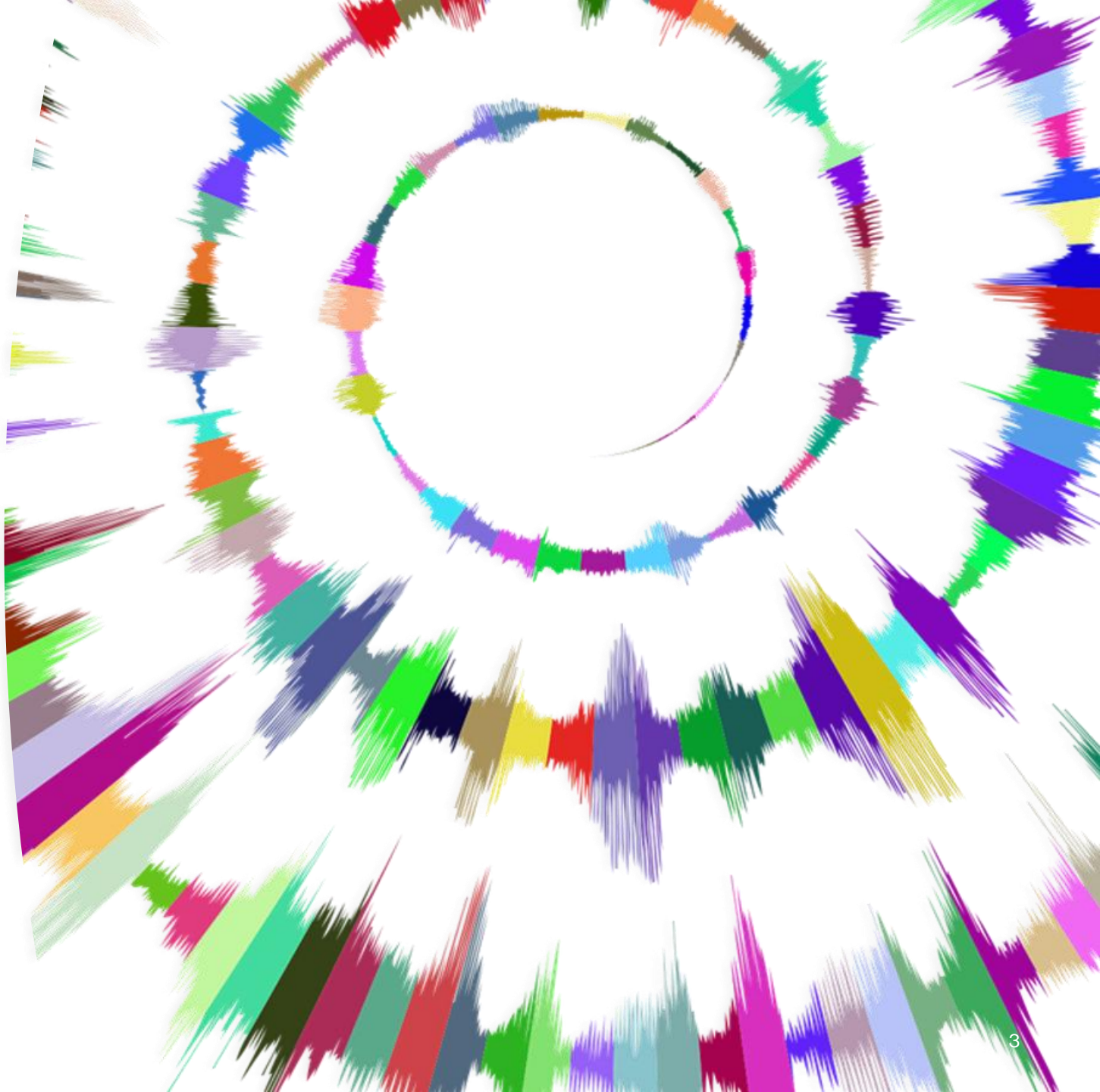
Hands-on Coding Session II
Music Visualization

Dr. Srisupang Thewsuwan

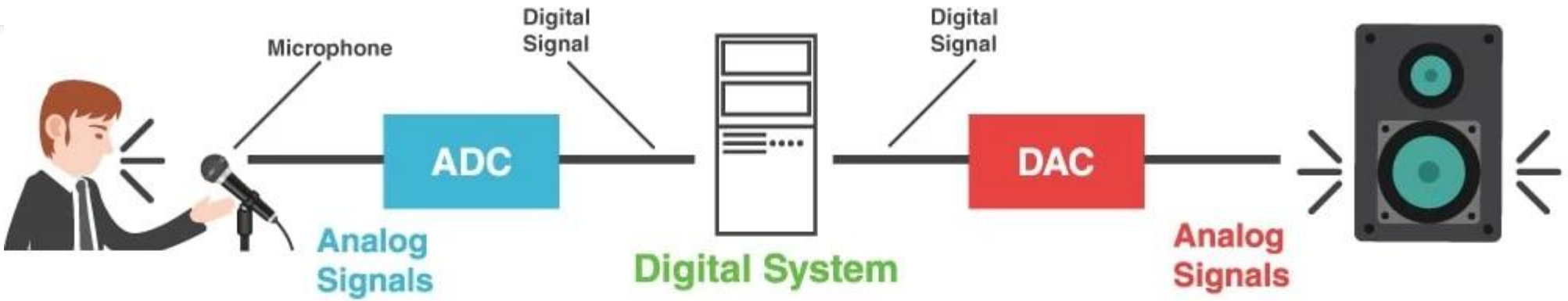
Introduction

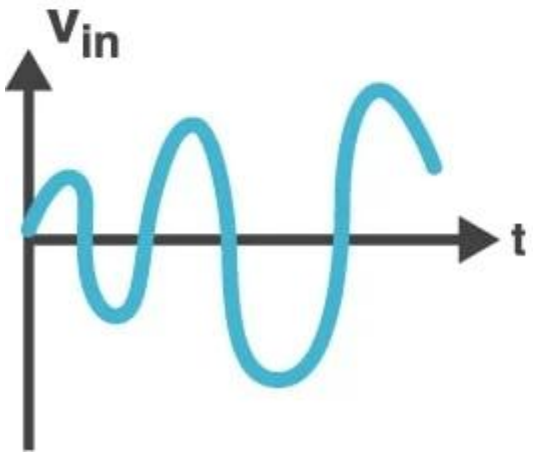
Music Visualization

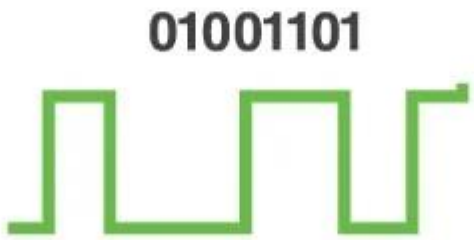
- The process of creating visual representations of audio signals, such as music.
- To provide a way to analyze and understand audio signals, and to create visually appealing representations of audio signals for artistic or entertainment purposes.

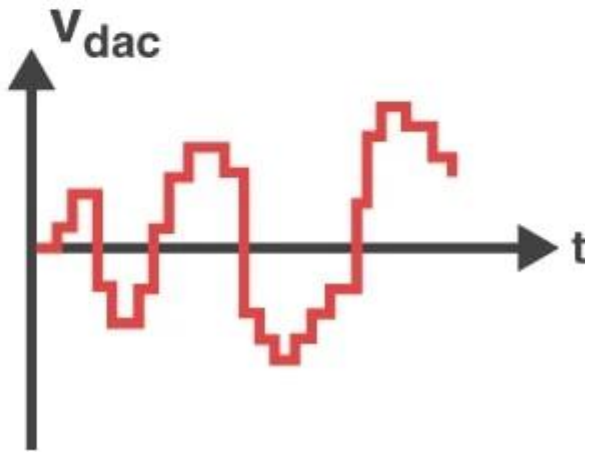


Audio signal processing

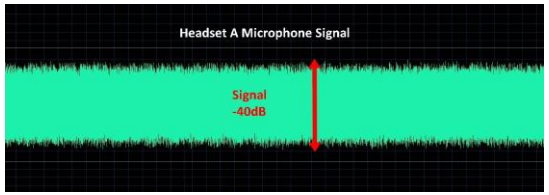


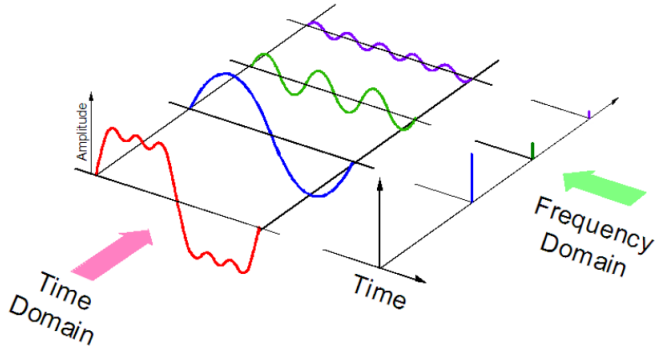


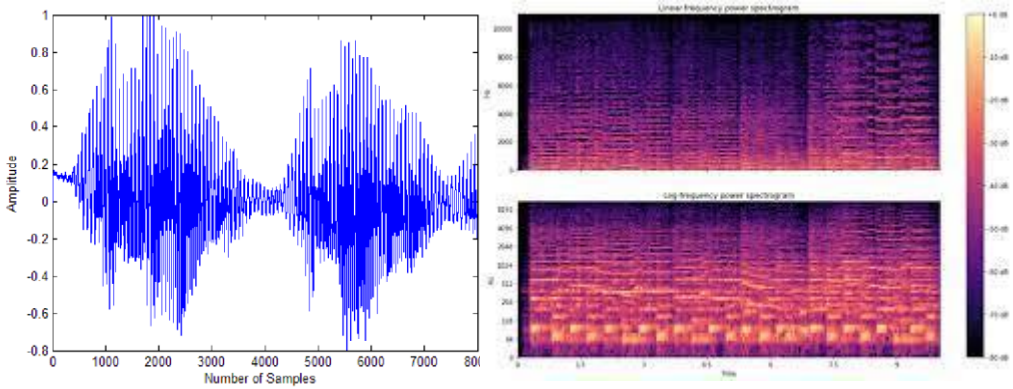




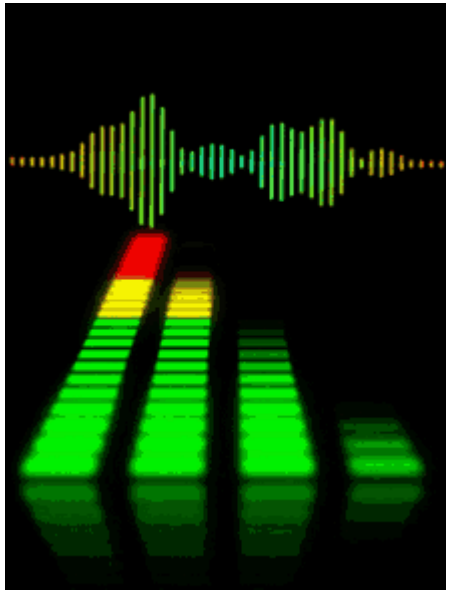
Audio input → Signal Processing → Data Representation



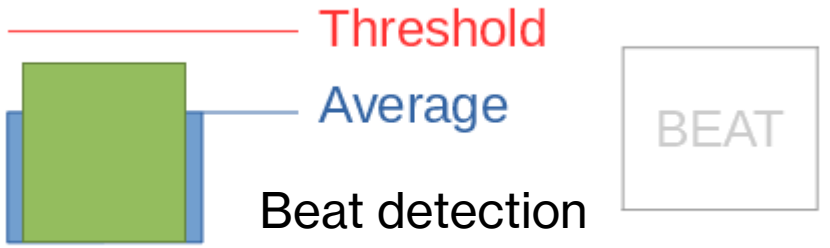




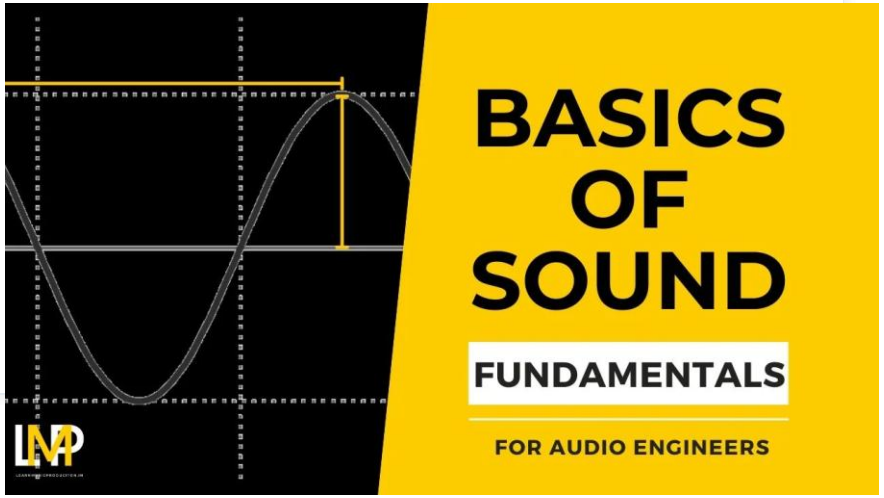
Workflow



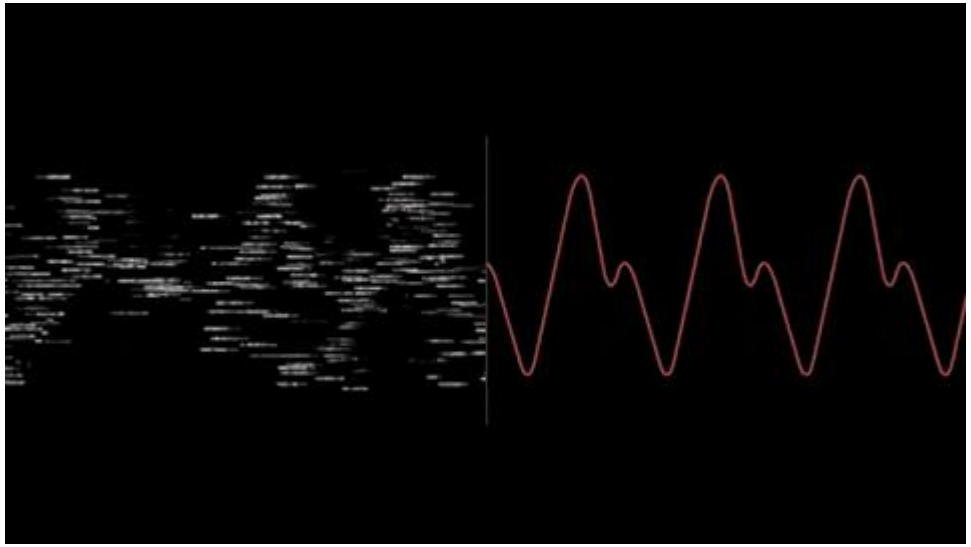
Visualization

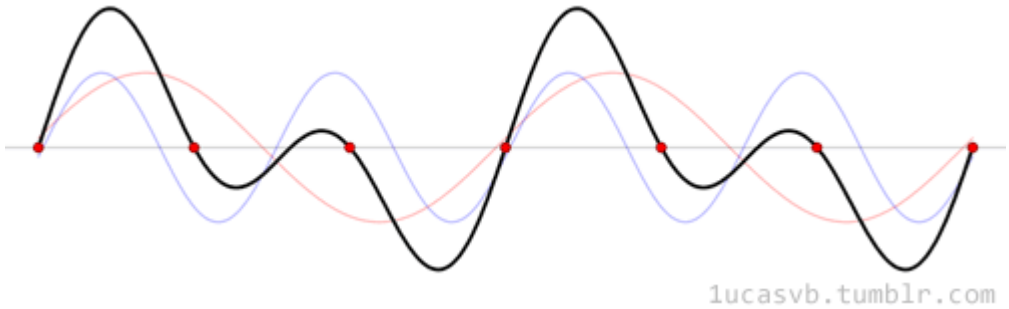


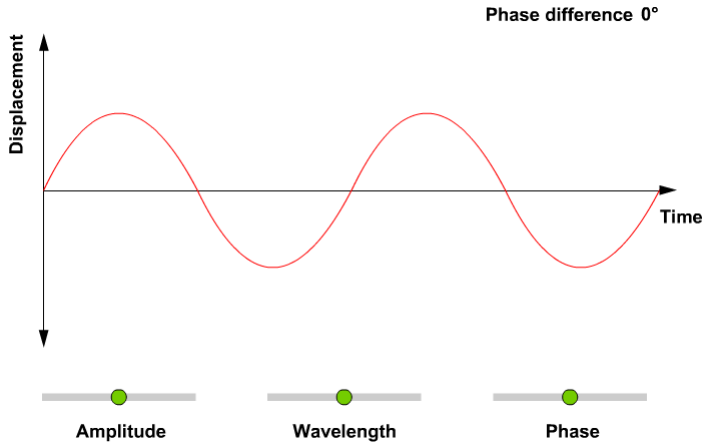
Significance in Audio Engineering



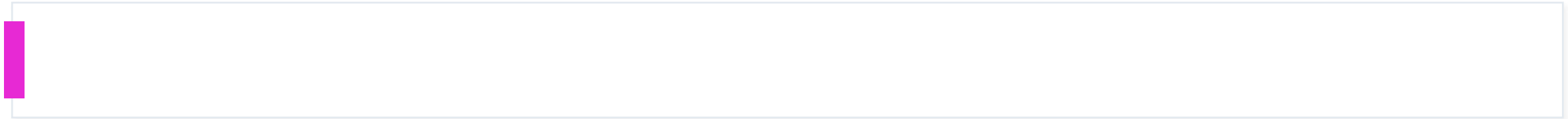
- Importance of visualizing audio signals to understand frequency content, structure, and changes over time.
- Applications in music production, sound design, and research.





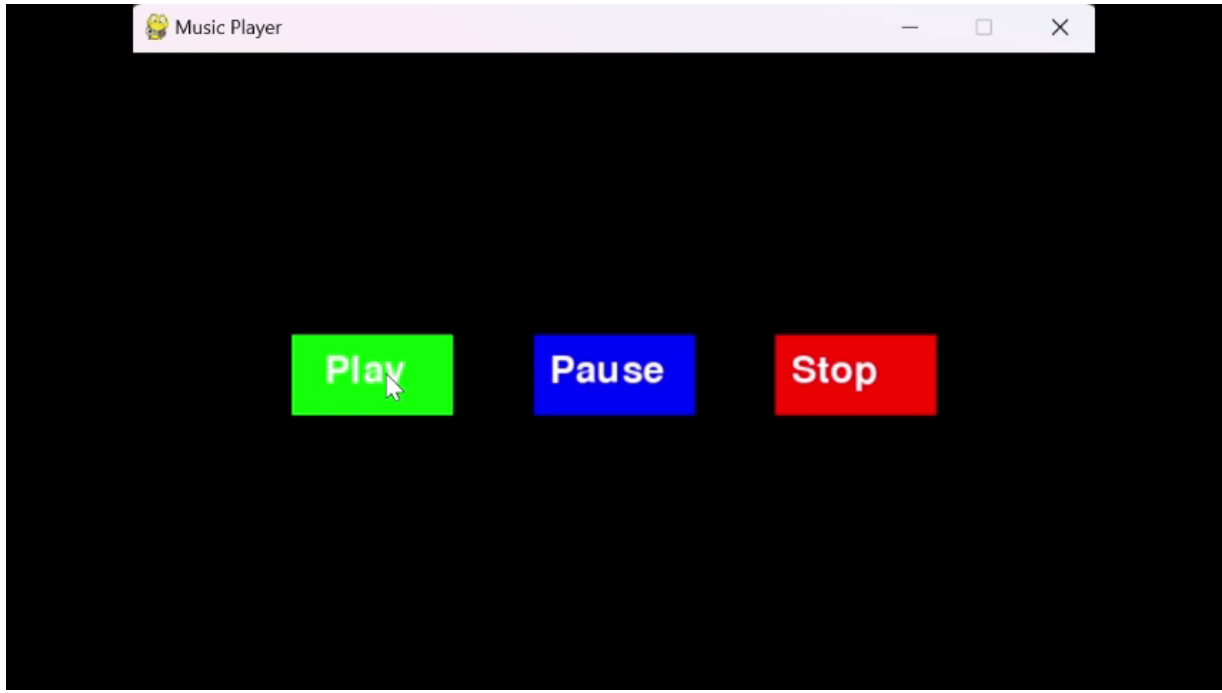


01- Create Music control buttons



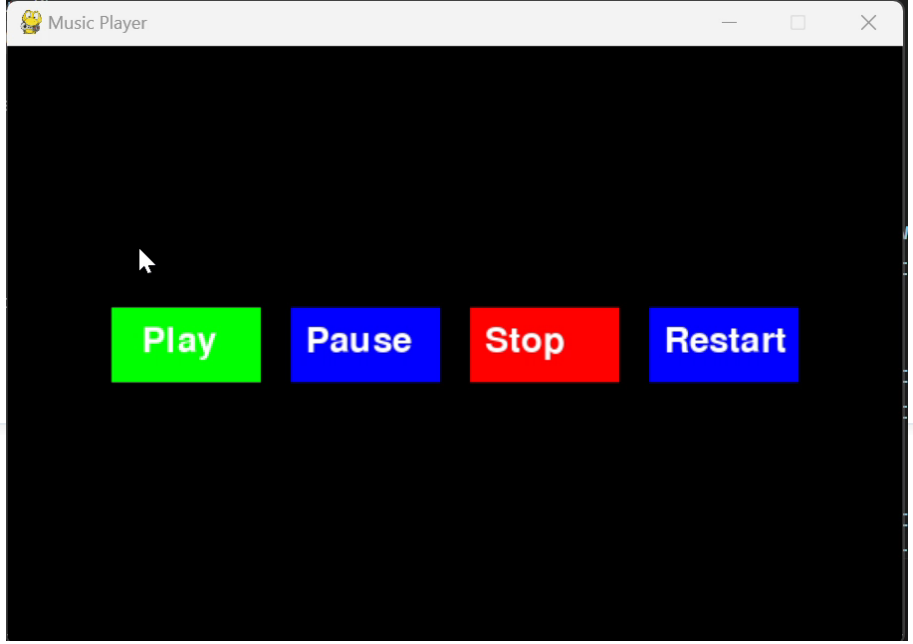
Task01-starter_music_playback

```
def draw_buttons():
    # Draw the play button
    pygame.draw.rect(screen, GREEN, play_button)
    play_text = font.render('Play', True, WHITE)
    screen.blit(play_text, (play_button.x + 20, play_button.y + 10))

    # Draw the pause button
    pygame.draw.rect(screen, BLUE, pause_button)
    pause_text = font.render('Pause', True, WHITE)
    screen.blit(pause_text, (pause_button.x + 10, pause_button.y + 10))

    # Draw the stop button
    pygame.draw.rect(screen, RED, stop_button)
    stop_text = font.render('Stop', True, WHITE)
    screen.blit(stop_text, (stop_button.x + 10, stop_button.y + 10))
```



Task01_Implement “Restart” Button

“Restart” Button



1. Add a Restart Button

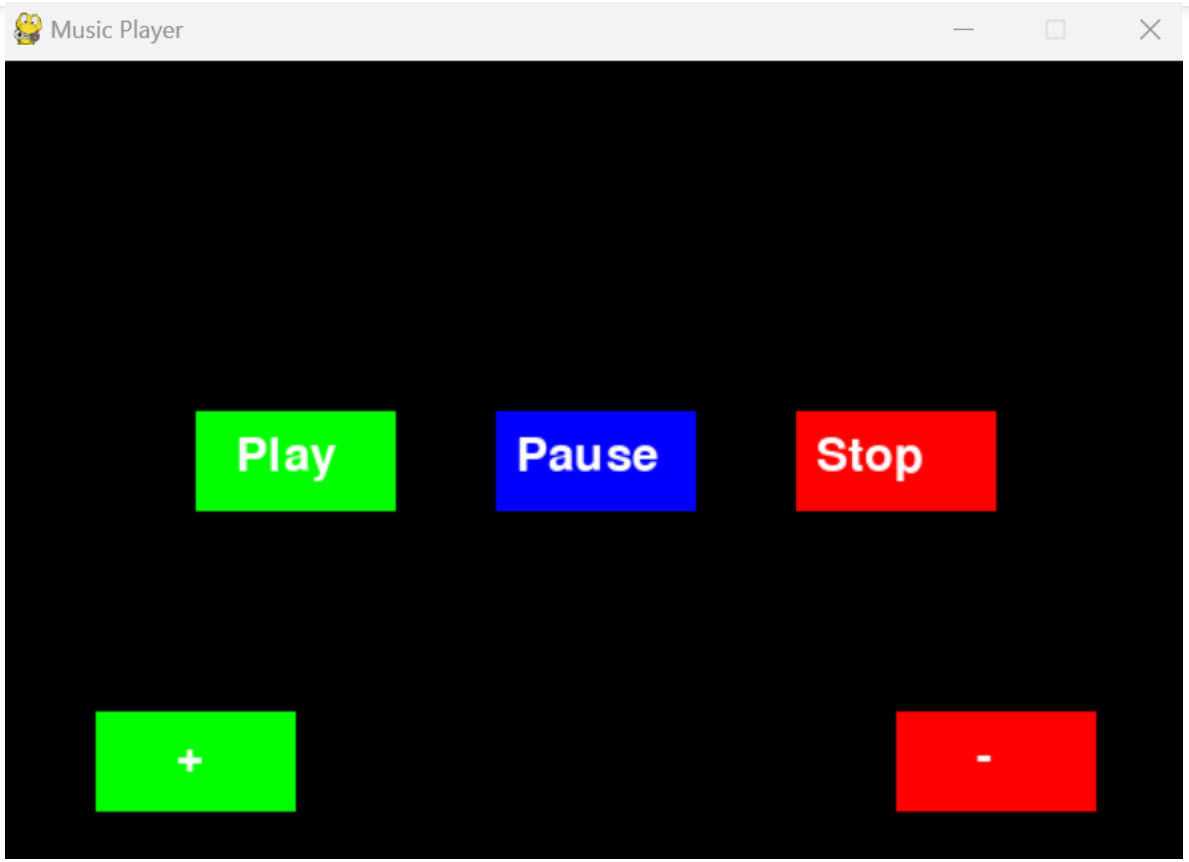
```
restart_button = pygame.Rect(4 * WIDTH // 5 - button_width // 2, HEIGHT // 2 - button_height // 2, button_width, button_height)
```

2. Draw the Restart Button
3. Implement the Restart Button Functionality

```
elif restart_button.collidepoint(event.pos):  # Check if Restart button is clicked
        pygame.mixer.music.play(start=0)  # Restart the music from the beginning
```

02-Volume Up and Down Buttons

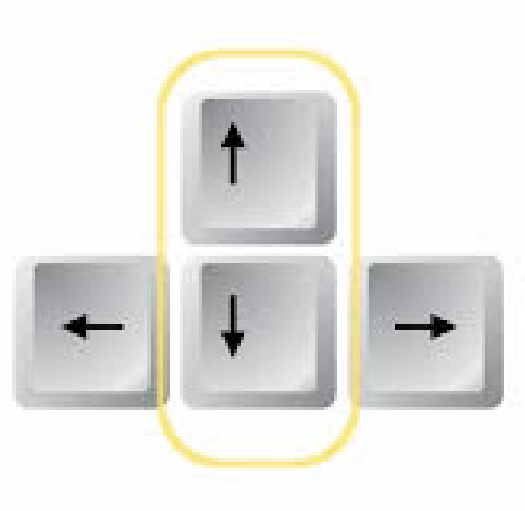
02-volumnButtons_music_playback



Task02-Adjust the Volume with Keyboard Keys

Task02-Adjust the Volume with Keyboard Keys

```
elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP:  # Increase volume
            current_volume = increase_volume(current_volume)
        elif event.key == pygame.K_DOWN:  # Decrease volume
            current_volume = decrease_volume(current_volume)
```



Task03: Create Volume Control Module

- Goal: Create a volume_control.py module that defines functions to increase and decrease the music volume. Integrate this module with the main PyGame music player script (main.py) to control the volume using keyboard inputs.

volume_control.py

```
import pygame

def increase_volume(current_volume, increment=0.1):
    new_volume = min(current_volume + increment, 1.0)
    pygame.mixer.music.set_volume(new_volume)
    return new_volume

def decrease_volume(current_volume, decrement=0.1):
    new_volume = max(current_volume - decrement, 0.0)
    pygame.mixer.music.set_volume(new_volume)
    return new_volume
```

main.py

```
import pygame
import sys
from volume_control import increase_volume, decrease_volume

# Initialize PyGame, set up the display, and load music as before

current_volume = pygame.mixer.music.get_volume()

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:  # Increase volume
                current_volume = increase_volume(current_volume)
            elif event.key == pygame.K_DOWN:  # Decrease volume
                current_volume = decrease_volume(current_volume)

    # Fill the screen, draw UI elements, etc.

    pygame.display.update()

pygame.quit()
sys.exit()
```

03-audioProcesswithNumpy

03-audioProcesswithNumpy

This normalization and scaling process allows the waveform to be visually represented in a way that maximally utilizes the available space

```
amplitude = np.interp(signal, (signal.min(), signal.max()), (0, HEIGHT))
```

Task04_Stop and Play Music with Keyboard Keys


```
elif event.type == pygame.KEYDOWN:
            # Stop music if the 's' key is pressed
        if event.key == pygame.K_s:
            pygame.mixer.music.stop()
         # Play music if the 'p' key is pressed
        if event.key == pygame.K_p:
            pygame.mixer.music.play()
```

04-fileMenu_music_playback

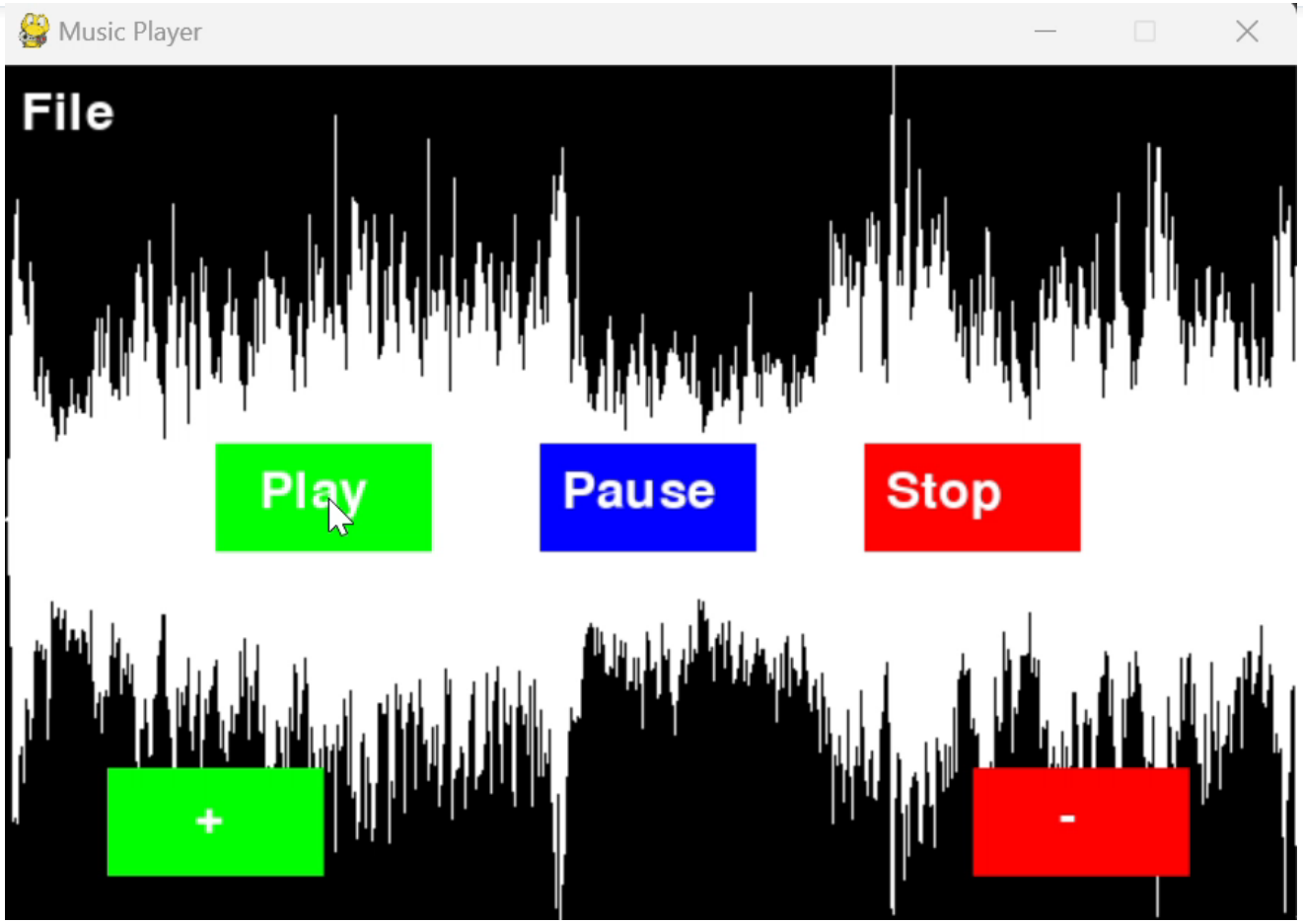
04-fileMenu_music_playback

import tkinter as tk

Task05_Create Music Player Module

Task05_Create Music Player Module

1. volume_control.py
2. audio_visualization.py
3. filemenu.py
4. main.py



Additional Resources

Music Visualizer by Beat Detection

1. Libraries Installation

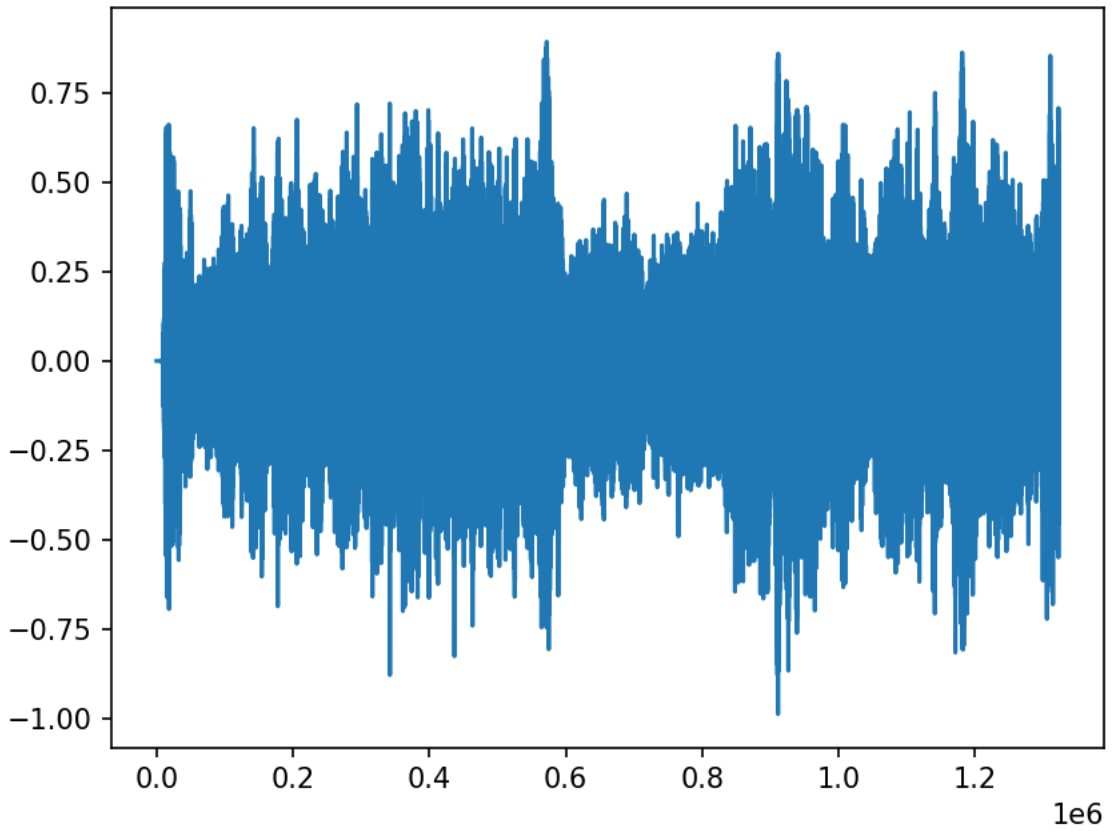
- Pygame
- Numpy
- Sounddevice
- soundfile
- Matplotlib
- Aubio
- Pydub
- librosa

pip3 install requirements.txt
Or
pip install requirements.txt

2. Audio visualization

```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
import soundfile as sf


# Load the audio file
filename = 'StarWars60.wav'
#data, samplerate = sd.read(filename)
data, samplerate = sf.read(filename, dtype='float32')

# Get the average amplitude of the audio data
average_amplitude = np.mean(np.abs(data))
print(average_amplitude)
# Plot the audio data
plt.plot(data)

# Show the plot
plt.show()
```

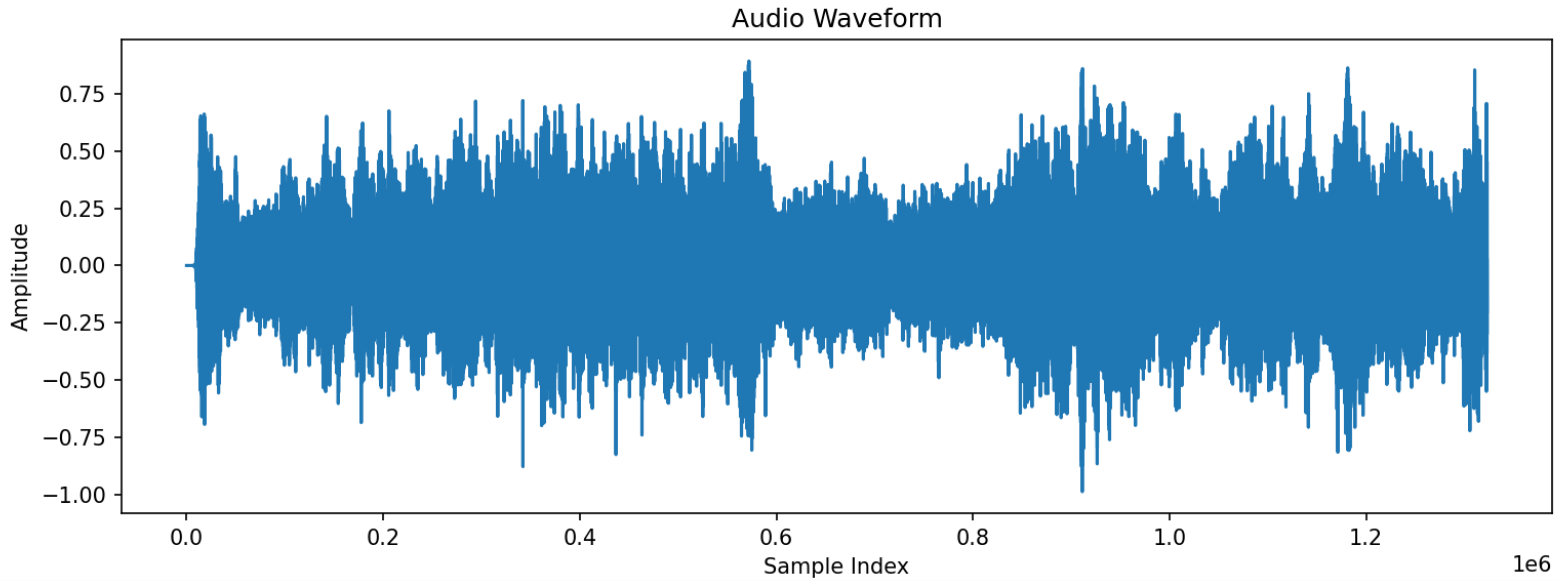
Displaying waveform



3. Pydub library

- Convert audio data to numpy array using pydub

```
from pydub import AudioSegment
import numpy as np
import matplotlib.pyplot as plt

def display_waveform(file_path):
    audio = AudioSegment.from_file(file_path)

    # Convert audio data to numpy array
    audio_data = np.array(audio.get_array_of_samples())
    audio_data = audio_data / (2**15)

    # Plot the audio data
    plt.figure(figsize=(12, 4))
    plt.plot(audio_data)
    plt.title('Audio Waveform')
    plt.xlabel('Sample Index')
    plt.ylabel('Amplitude')
    plt.show()

# Replace "your_audio_file.mp3" with the path to your audio file
file_path = "StarWars60.wav"
display_waveform(file_path)
```

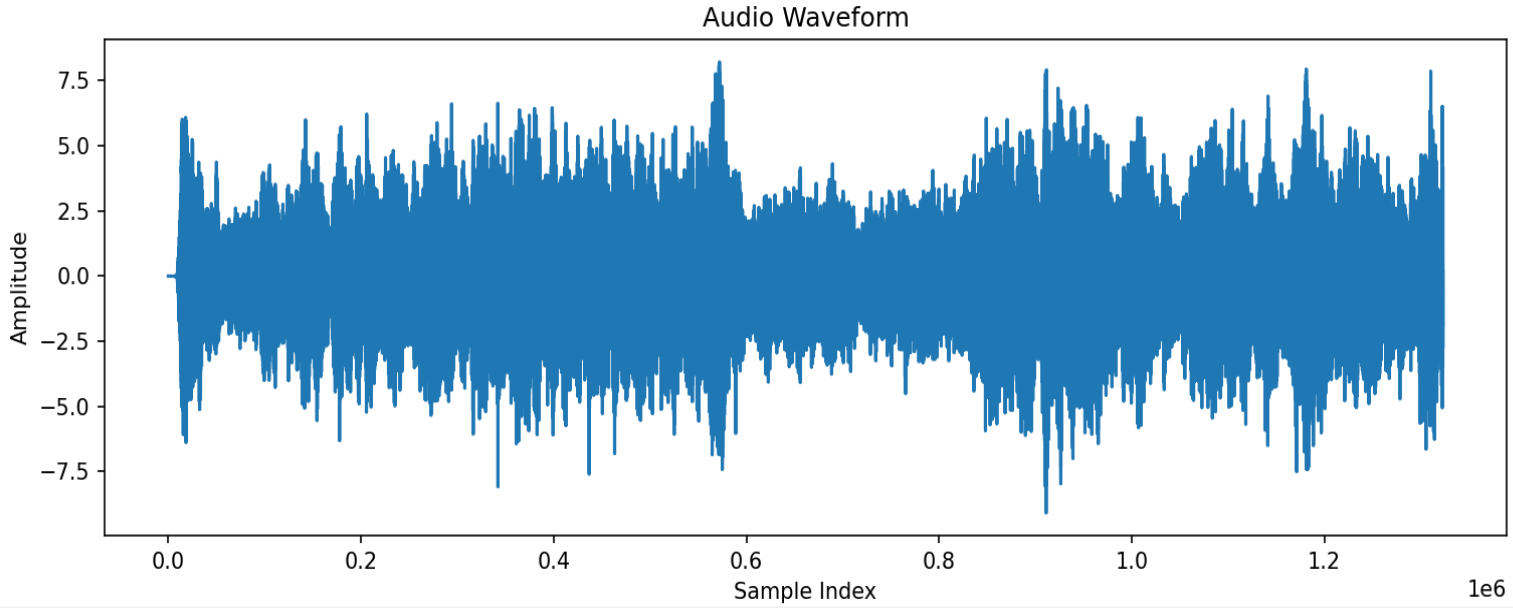


3. Pydub library

- Convert audio data to numpy array using pydub

```
audio_data = audio_data / (np.mean(np.abs(audio_data)))
```

```
from pydub import AudioSegment
import numpy as np
import matplotlib.pyplot as plt

def display_waveform(file_path):
    audio = AudioSegment.from_file(file_path)

    # Convert audio data to numpy array
    audio_data = np.array(audio.get_array_of_samples())
    audio_data = audio_data / (2**15)

    # Plot the audio data
    plt.figure(figsize=(12, 4))
    plt.plot(audio_data)
    plt.title('Audio Waveform')
    plt.xlabel('Sample Index')
    plt.ylabel('Amplitude')
    plt.show()

# Replace "your_audio_file.mp3" with the path to your audio file
file_path = "StarWars60.wav"
display_waveform(file_path)
```



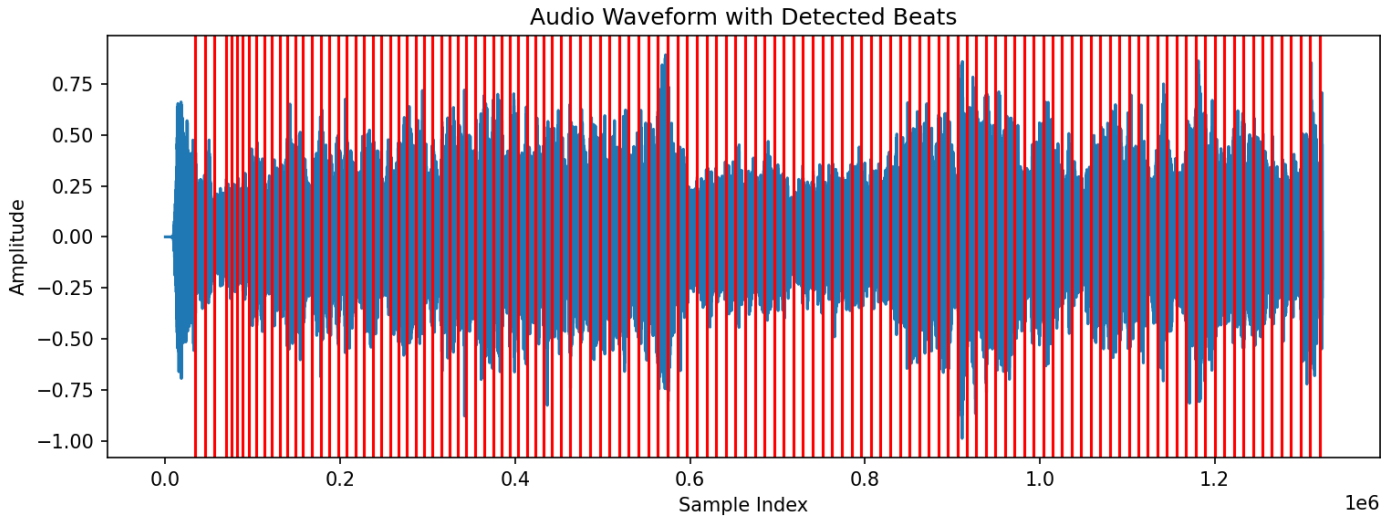
4. Aubio library for beat detection

FFT Flux formula

Spectral Flux method, which measures the difference in energy between consecutive short-time spectral frames.

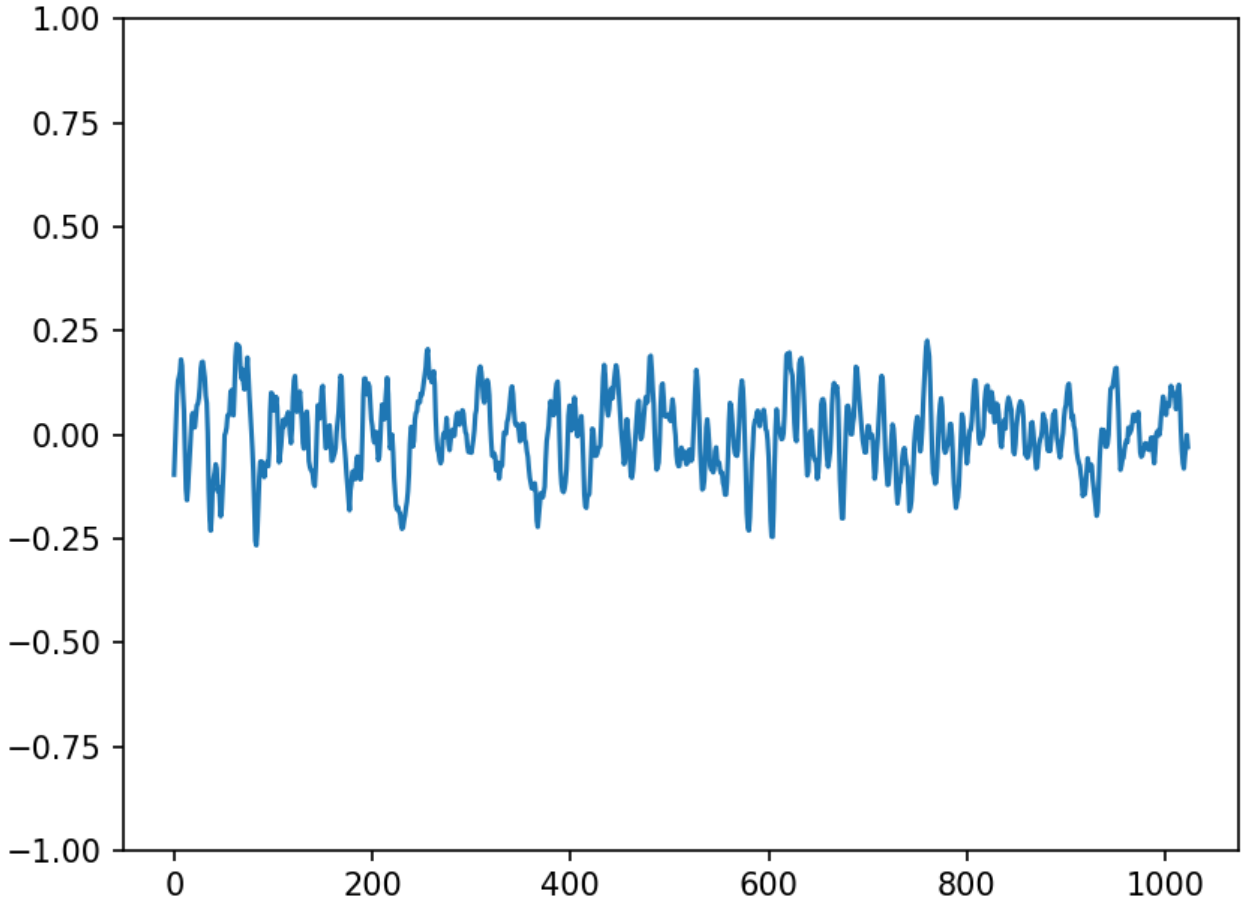
$$x_n \xrightarrow{\text{window+FFT}} X_k \xrightarrow{\text{magnitude}} a_k$$

$$Flux(m) = \frac{\sum_k |a_k[m] - a_k[m-1]|}{\sum_k a_k[m] + a_k[m-1]}$$




```python
import numpy as np
import matplotlib.pyplot as plt
from pydub import AudioSegment
from aubio import source, tempo


def display_waveform_with_beats(file_path):
    audio = AudioSegment.from_file(file_path)

    # Convert audio data to numpy array
    audio_data = np.array(audio.get_array_of_samples())
    audio_data = audio_data / (2 ** 15)

    # Set up beat detection
    win_s = 512
    hop_s = win_s // 2
    samplerate = audio.frame_rate
    s = source(file_path, samplerate, hop_s)
    o = tempo("default", win_s, hop_s, samplerate)

    # Collect beat positions
    beats = []
    while True:
        samples, read = s()
        is_beat = o(samples)
        if is_beat:
            beats.append(o.get_last_s())
        if read < hop_s:
            break

    # Plot the audio data
    plt.figure(figsize=(12, 4))
    plt.plot(audio_data)

    # Plot the detected beats
    for beat in beats:
        plt.axvline(x=beat * samplerate, color='r', linestyle='-')

    plt.title('Audio Waveform with Detected Beats')
    plt.xlabel('Sample Index')
    plt.ylabel('Amplitude')
    plt.show()


# Replace "your_audio_file.mp3" with the path to your audio file
file_path = "StarWars60.wav"
display_waveform_with_beats(file_path)
```

5. Audio Stream



Tasks

Task1 Change the color of the bars in the visualization:

- You can use the `color` parameter in the `ax.bar()` function to specify a color of your choice.

Task2 Add a grid to the visualization:

- You can use the `ax.grid()` function to add a grid with custom line style, color, and transparency.

Task3 Add labels to the axes:

- They can use the ax.set_xlabel() and ax.set_ylabel() functions to add labels to the axes.

Task4 Modify the window function:

- Instead of using Hanning window, you can explore other window functions like Hamming, Blackman, or Kaiser and see how the choice of the window function affects the visualization.

Task5 Adjust the blocksize and FFT size:

- You can experiment with different values for the blocksize and the size of the FFT to see how these choices affect the visualization's resolution and responsiveness.