Ήχος στο Διαδίκτυο

Web Audio API
Audio Sources and Waveform Rendering

Χρυσούλα Αλεξανδράκη
Τμήμα Μουσικής Τεχνολογίας και Ακουστικής
Ελληνικό Μεσογειακό Πανεπιστήμιο

Audio Sources

- Nodes that GENERATE or INPUT sound into the audio graph.
  - Only **source nodes can START sound**
  - Everything else (Gain, Delay, Filter, etc.) just processes it
- Five types:

Source Type	**Represents**
OscillatorNode	Generated waveform
AudioBufferSourceNode	Loaded audio data
MediaElementAudioSourceNode	<audio>/<video> element
MediaStreamAudioSourceNode	Microphone / live input
MediaStreamTrackAudioSourceNode	Single track from stream

MediaStreamAudioSourceNode

- To play audio from file
- Creating

```
const audio = new Audio('../audio/disco0.mp3');
audioSrc = ctx.createMediaElementSource(audio);
audio.play();
```

- In order to hear any sound, you have to call play() on the audio element
  - No restriction on calling play/stop/pause
  - Audio is an element of the HTML DOM API

OscillatorNode

- To generate sample data
- Creating

```
const oscil = ctx.createOscillator();
  src.frequency.value = 110;
  src.type = 'sawtooth';
  src.start();
```

- NB: Start can only be called once!
  - Instead of starting/stopping the oscillator, choose to resume/suspend the AudioContext

AudioBufferSourceNode - Create

- To playback an array with your own samples
- Create

```
src = ctx.createBufferSource();
const dur = 10; //sec
const myBuffer = ctx.createBuffer(2, ctx.sampleRate*dur, ctx.sampleRate);
loadBuffer(); //a func load values in the interval [-1, 1]
src.buffer = myBuffer;
src.start();
```

AudioBufferSourceNode - Load

- The following computes a semitone at 440 Hz

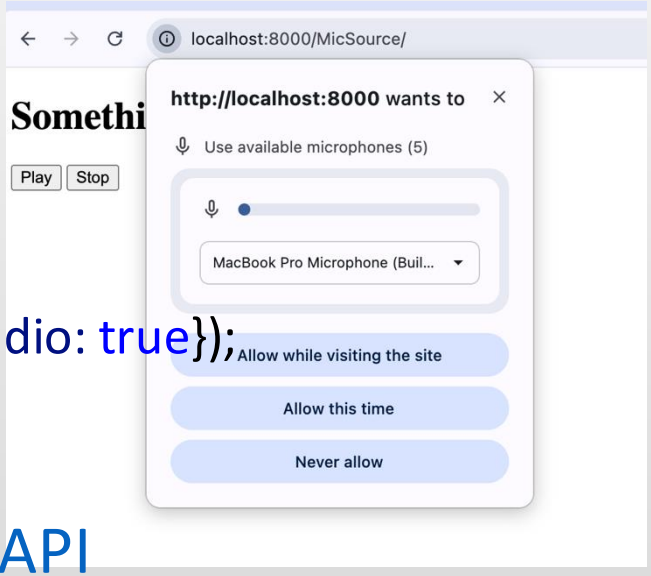
```
function loadBuffer() {
    f = 440;
    for (let ch = 0;ch< myBuffer.numberOfChannels;ch++) {
        let channelBuffer = myBuffer.getChannelData(ch
        for (let i=0;i<myBuffer.length;i++ ) {
                    channelBuffer[i] =  Math.random() * 2 - 1;

        }
    }
}
```

- Check Math JS Built-in object for more buffer populating functions

MediaStreamAudioSourceNode - create

- Create:

```
async function startMic() {
    ctx = new AudioContext();
    // Get microphone stream
    const stream = await          navigator.mediaDevices.getUserMedia({audio: true});
    src = ctx.createMediaStreamSource(stream);
}
```



- The function getUserMedia() is part of the Media Devices API
  - Does not have to do with Web Audio API
- Stream is created asynchronously
  - Waits for the user to allow access to microphone

MediaStreamAudioSourceNode - use

- You need to wait for the source to be created before you connect the nodes in the audio graph!!
- For example:
  - You need to create function createAudioGraph(); and call it within the async function, i.e. after the mocrophone stream has been assigned

AnalyserNode

- Provides time-domain or frequency-domain windows of a waveform
  - Their length is defined by the fftSize parameter
- Create

```
analyser = ctx.createAnalyser();
analyser.fftSize = 2048;
```

- Connect its input to a GainNode and its output to the AudioContext destination

```
// Build audio graph
src.connect(gain);
gain.connect(analyser);
analyser.connect(ctx.destination); // live monitoring
```

- To read waveform data initiate an int array (8-bits → values: 0-256)

```
bufferLength = analyser.fftSize;
dataArray = new Uint8Array(bufferLength);
```

- And fill it with time or frequency domain data with unsigned byte array (8-bit -> 0 -255)

```
getByteTimeDomainData(dataArray);
getByteFrequencyData(dataArray);
```

Waveform Graphic Rendering

- Use the canvas HTML Element

```
<canvas id="scope" width="600" height="200"></canvas>
```

- In JS, a draw() function is called to draw the canvas

```
// --- Canvas setup ---
const canvas = document.getElementById("scope");
const c = canvas.getContext("2d");
let bufferLength, dataArray;
draw();
```

- Inside draw() function the following as the first command.

```
requestAnimationFrame(draw);
```

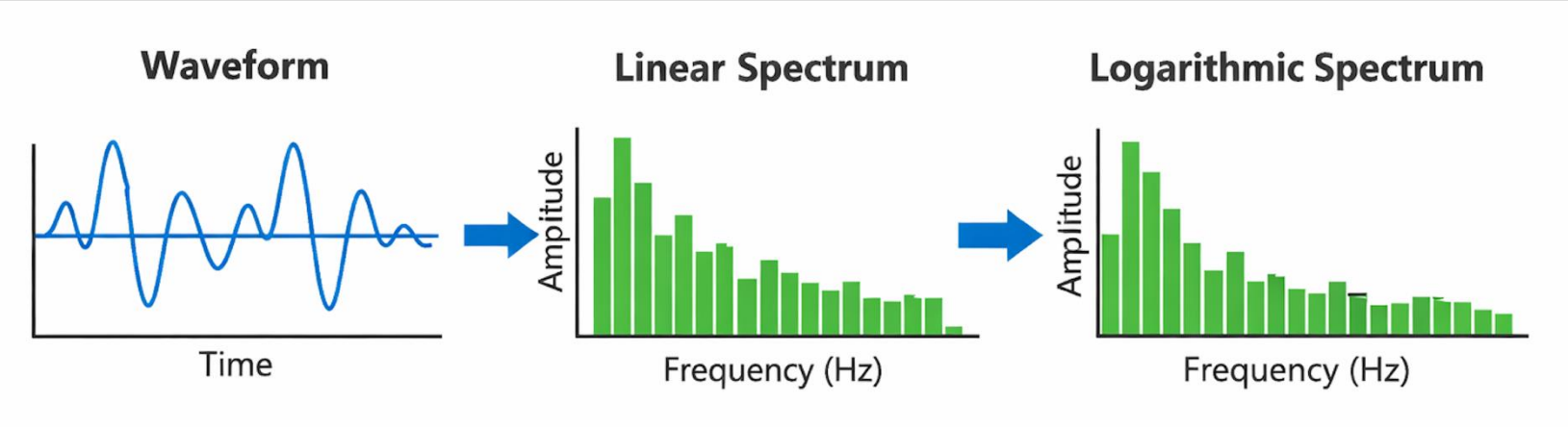
- This tells the browser to call draw() repeatedly at a standard rate (eg. 60Hz) to render the animation frames. To initiate you first call it explicitly in your code and then use it as callback.

The draw() function

```
function draw() {
        requestAnimationFrame(draw);

        if (analyser) analyser.getByteTimeDomainData(dataArray);

        c.fillStyle = "black";
        c.fillRect(0, 0, canvas.width, canvas.height);
        c.lineWidth = 2;
        c.strokeStyle = "lime";
        c.beginPath();
        const sliceWidth = canvas.width / bufferLength;
        let x = 0;
        for (let i = 0; i < bufferLength; i++) {
                const v = dataArray[i] / 128.0; // 0–255 → 0–2
                const y = v * canvas.height / 2;
                if (i === 0) c.moveTo(x, y);
                else c.lineTo(x, y);
                x += sliceWidth;
        }
        c.lineTo(canvas.width, canvas.height / 2);
        c.stroke();
}
```

FFT Spectrum

- FFT: reversible mathematical process to check how much amplitude is contained in each frequency (bin)
- Log Spectrum because
  - allows to show more detail (than linear spectrum) in the low frequencies
  - It resembles psychoacoustics both for amp and for freq



The drawSpectrum() function

```
function drawSpectrum() {
    requestAnimationFrame(drawSpectrum);
    if (analyser) analyser.getByteFrequencyData(specData);
    c2.fillStyle = "black";
    c2.fillRect(0, 0, canvas2.width, canvas2.height);
    const barWidth = 2; //canvas.width / bufferLength;
    let x = 0;
    for (let i = 0; i < bufferLength; i++) {
                const barHeight = specData[i]/255 * canvas2.height;
                c2.fillStyle = "lime";
                c2.fillRect(x, canvas2.height - barHeight, barWidth, barHeight);
                x += barWidth + 1; //leave an empty line
    }}
```

The drawLogSpectrum() function

```
function drawLogSpectrum() {
        requestAnimationFrame(drawLogSpectrum);

        if (analyser) analyser.getByteFrequencyData(dataArray);

        c.fillStyle = "black";
        c.fillRect(0, 0, canvas.width, canvas.height);
        const nyquist = ctx.sampleRate / 2;
        for (let i = 1; i < bufferLength; i++) {
                const freq = i * nyquist / bufferLength;
                // Log position (20 Hz – Nyquist)
                const minLog = Math.log10(20);
                const maxLog = Math.log10(nyquist);
                const x = (Math.log10(freq) - minLog) / (maxLog - minLog)* canvas.width;
                const value = dataArray[i];
                const height = value / 255 * canvas.height;
                c.fillStyle = "lime";
                c.fillRect(x, canvas.height - height, 2, height);
        }
}
```

Methods of HTML Canvas

- Drawing a rectangle
  - The **fillRect()** method draws a black rectangle with a top-left corner at position 20,20. The rectangle is 150 pixel wide and 100 pixels high.

```
const myCanvas =
document.getElementById("myCanvas");
const ctx = myCanvas.getContext("2d");
ctx.fillRect(20, 20, 150, 100);
```

- Drawing a lines

```
ctx.beginPath();
ctx.moveTo(20, 20);
ctx.lineTo(20, 100);
ctx.lineTo(70, 100);
ctx.stroke();
```