Web Audio API

Was JavaScript designed to build a modular synth

23/03/2023

Pierre Poliakoff

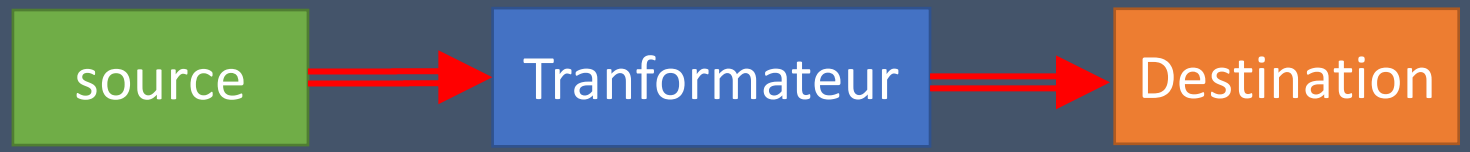
The modular Synth

Développé par Robert Moog en 1964
Formé de modules

- LFO
- VCO
- ADSR
- Sequencer

Un module peut être

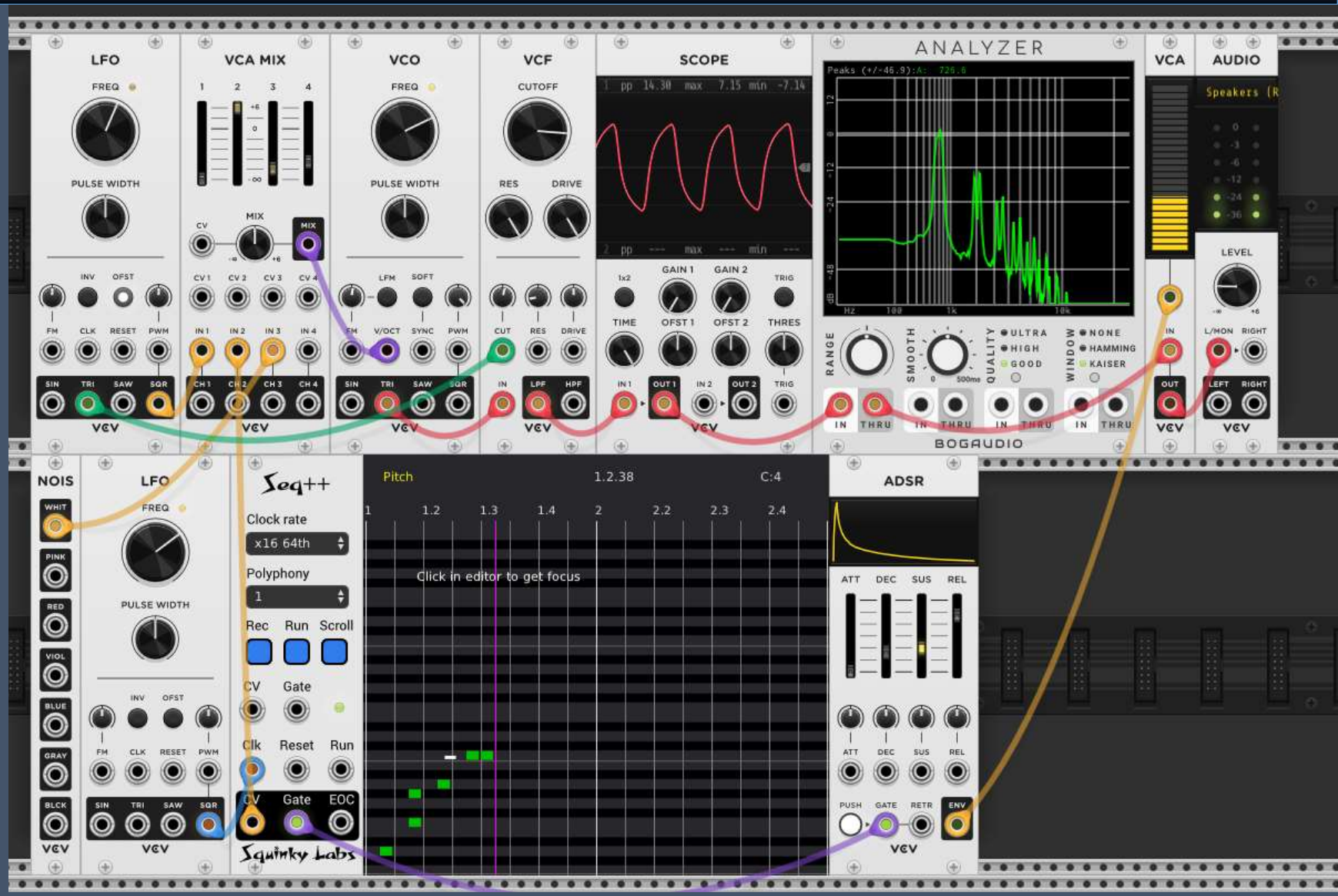
- Une source (génère un signal)
- Une destination (amplificateur + Haut parleurs, enregistreur, visualiseur,…)
- Un module intermédiaire qui transforme le signal (filtre, mixeur,…)



VCV Rack2

Simulateur de synthétiseur modulaire

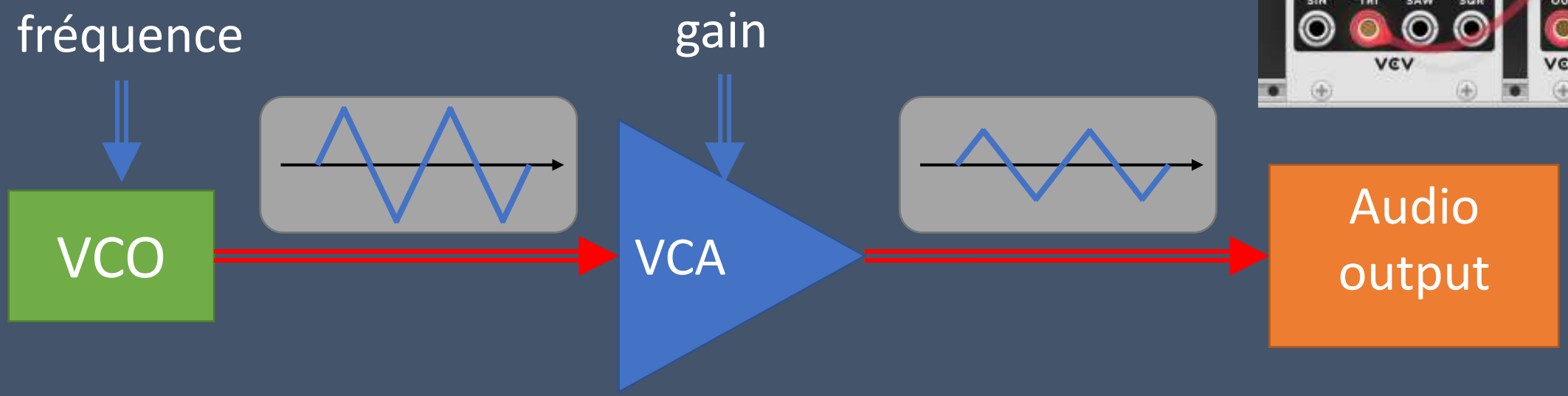
- Open source
- Dispose de centaines de modules



new_9_full.vcv

VCO + VCA


VCO
FREQ
PULSE WIDTH
LFM
SOFT
FM
V/OCT
SYNC
PWM
SIN
TRI
SAW
SQR
VCV
VCA
IN
OUT
VCV
AUDIO
Speakers
LEVEL
L/MON
RIGHT
LEFT
RIGHT
VCV


fréquence
gain
VCO
VCA
Audio output

Javascript Web audio API

- https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

Les objets les plus importants de la Web Audio API

- AudioContext
- AudioNode
- AudioParam

AudioContext

- C’est l’équivalent du rack vide avec une sortie audio
- Il crée les AudioNodes

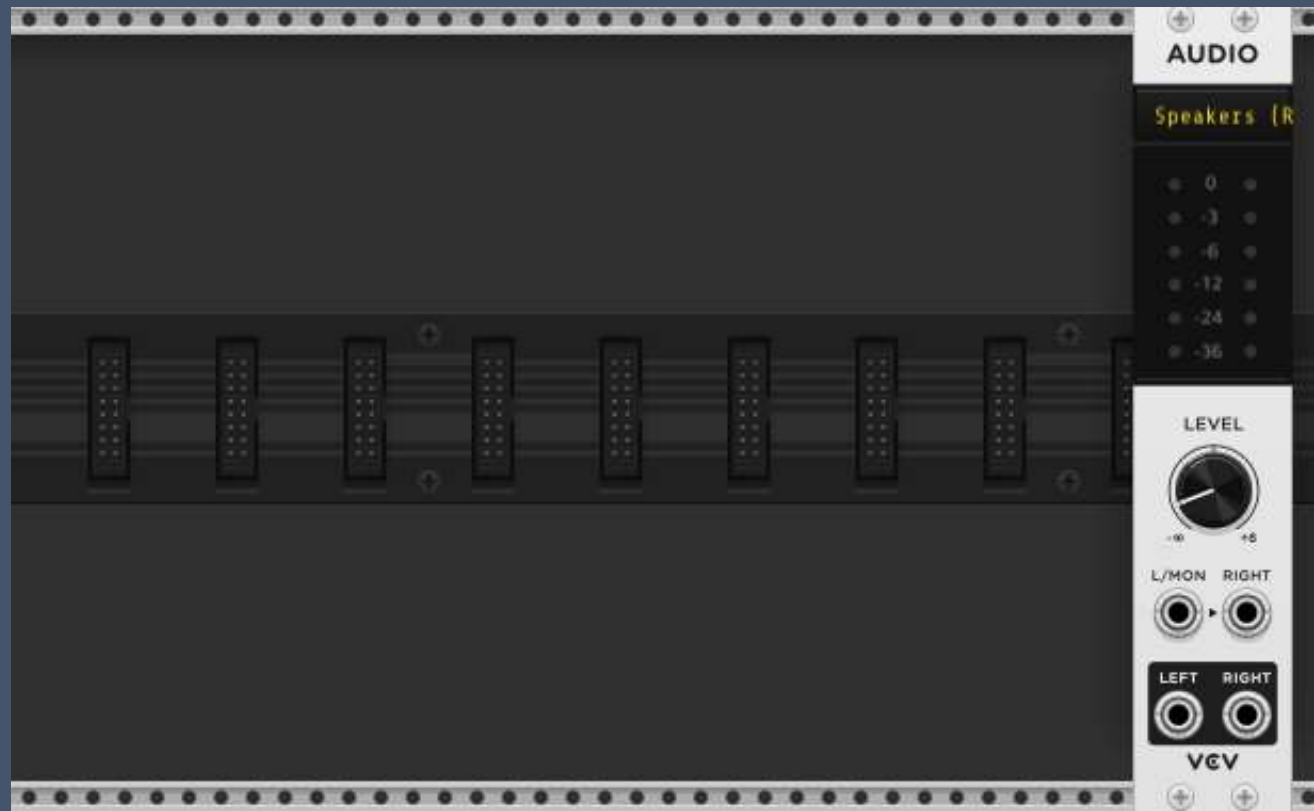
Attributs

- destination
- currentTime

Méthodes

- suspend()
- resume()
- createGain()
- createAnalyzer()
- createBuffer()
- createBufferSource()
- createBiquadFilter()
- …

Attention: il faut une « user gesture » pour pouvoir activer un AudioContext



AudioContext : code

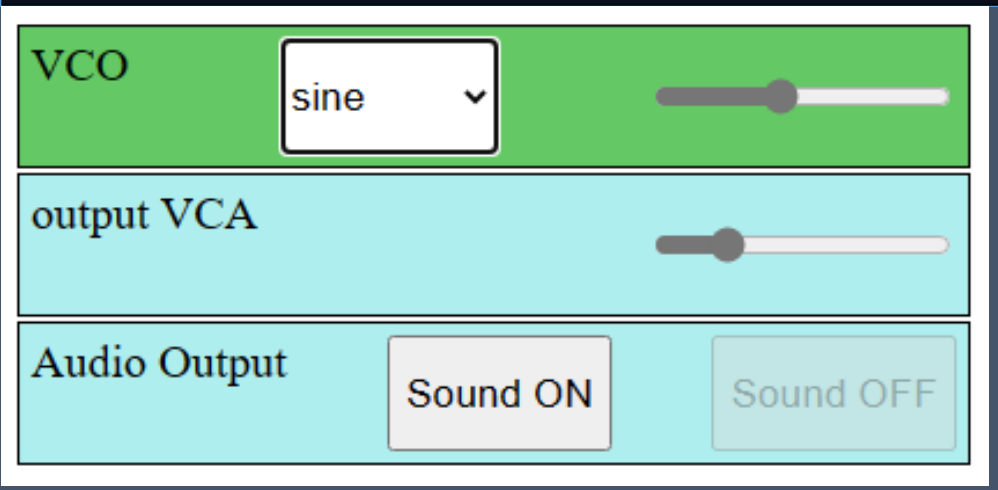
```
const audio_ctx = new AudioContext();
const audio_output = audio_ctx.destination;
audio_ctx.suspend();

const button_on = document.getElementById("button_on");
const button_off = document.getElementById("button_off");

button_on.addEventListener("click", (e) => {
    button_off.disabled = false;
    button_on.disabled = true;
    audio_ctx.resume();
});

button_off.addEventListener("click", (e) => {
    button_on.disabled = false;
    button_off.disabled = true;
    audio_ctx.suspend();
});
```

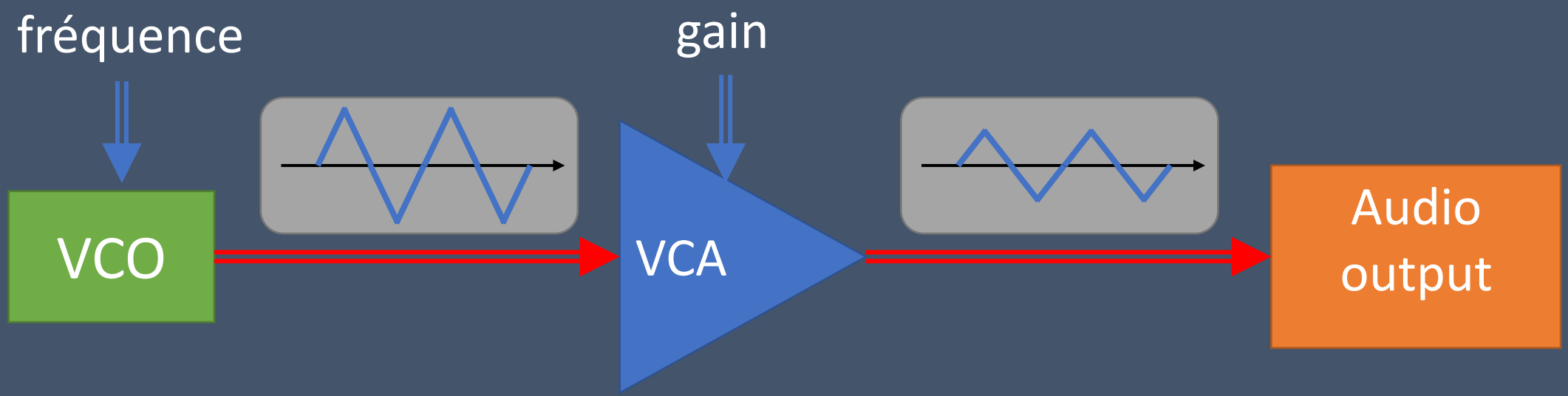
Les audioNodes

- C’est l’équivalent des modules
- Il y a des sources, des transformateurs et des destination
- On peut les connecter entre eux

  AudioNode.connect(destinationNode)
- Il en existe de très nombreux modèles
  - AnalyserNode
  - GainNode
  - OscillatorNode
  - AudioBufferSourceNode

Demo 1: VCO + VCA





AudioNodes: code

```
// create the tone generator
const osc1 = audio_ctx.createOscillator();
osc1.type = 'sine'; // values: sine, square,sawtooth,triangle,custom
osc1.frequencysetValueAtTime(600, audio_ctx.currentTime);

//create a output VCA
const output_vca = audio_ctx.createGain();
output_vca.gain.setValueAtTime(50, audio_ctx.currentTime);

//====================== Connect the nodes ========================
osc1.connect(output_vca);
output_vca.connect(audio_output);


osc1.start();
```

Les Audioparams

- C’est l ’équivalent de tout ce qui est contrôlé par des Control Voltage
- Définir la valeur

  `AudioParam.value=…`

- Définir la valeur à un certain moment

  `AudioParam.setValueAtTime()`

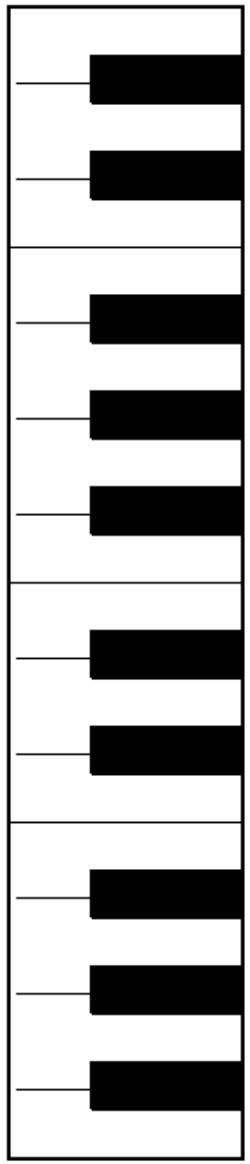
- Définir la valeur à un certain moment selon une courbe

  `AudioParam.linearRampToValueAtTime()`
  `AudioParam.exponentialRampToValueAtTime()`

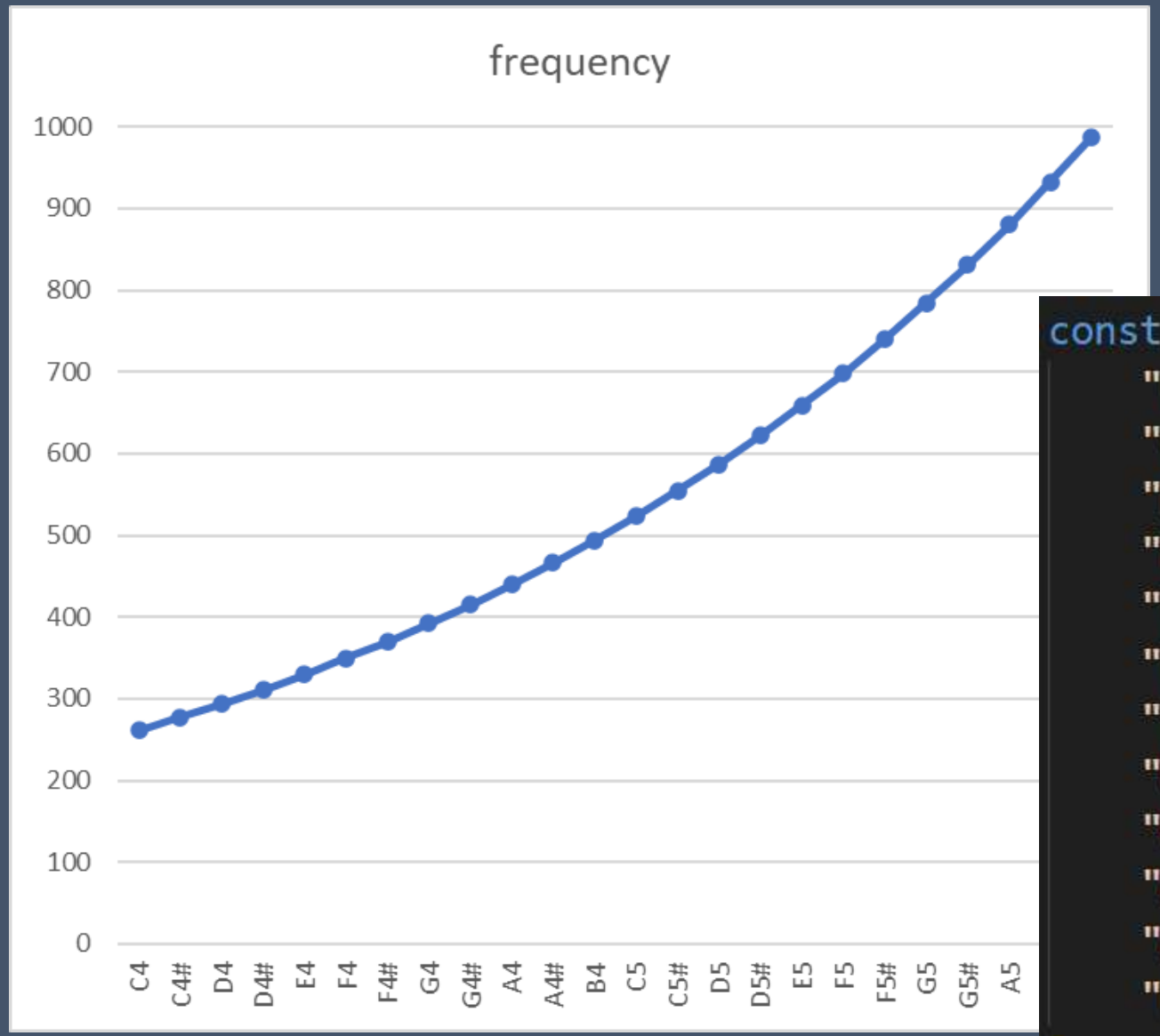
- Contrôler la valeur par un signal audio

  `myNode.connect(myAudioparam)`

Les notes de musique (Fréquence)



note	frequency
C4	261.63
C4#	277.18
D4	293.66
D4#	311.13
E4	329.63
F4	349.23
F4#	369.99
G4	392
G4#	415.3
A4	440
A4#	466.16
B4	493.88
C5	523.25
C5#	554.37
D5	587.33
D5#	622.25
E5	659.25
F5	698.46
F5#	739.99
G5	783.99
G5#	830.61
A5	880
A5#	932.33
B5	987.77




```
const noteFrequency = {
    "C4": 261.63,
    "C4#": 277.18,
    "D4": 293.66,
    "D4#": 311.13,
    "E4": 329.63,
    "F4": 349.23,
    "F4#": 369.99,
    "G4": 392,
    "G4#": 415.30,
    "A4": 440,
    "A4#": 466.16,
    "B4": 493.88,
};
```

Un générateur de notes aléatoires (VCV)

- Le LFO génère 5 pulses par secondes
- Le random Générateur génère une tension aléatoire à chaque pulse du LFO
- Le 1er VCA limite la tension de sortie du random generateur
- Le Quantizer convertit la tension en tension représentant des notes
- Le VCO convertit la tension en notes

Un générateur de notes aléatoires (Javascript)

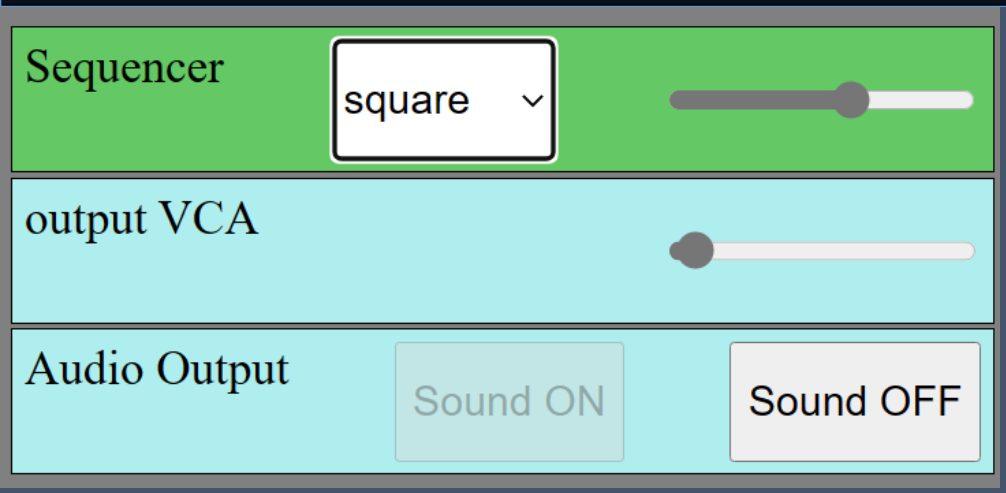
```
const scale = ["C4", "D4", "E4", "F4", "G4", "A4", "B4"];

nextNote(0);

function nextNote() {
    const osc = new OscillatorNode(audio_ctx, {
        frequency: noteFrequency[scale[Math.floor(Math.random() * scale.length)]],
        type: sequencer_panel.select.value
    });
    const duration = (0.01 + (100 - sequencer_panel.slider.value) / 90);

    osc.connect(output_vca);
    osc.addEventListener("ended", () => {
        nextNote();
    }, { once: true });

    const now = audio_ctx.currentTime;
    osc.start(now);
    osc.stop(now + duration);
}
```

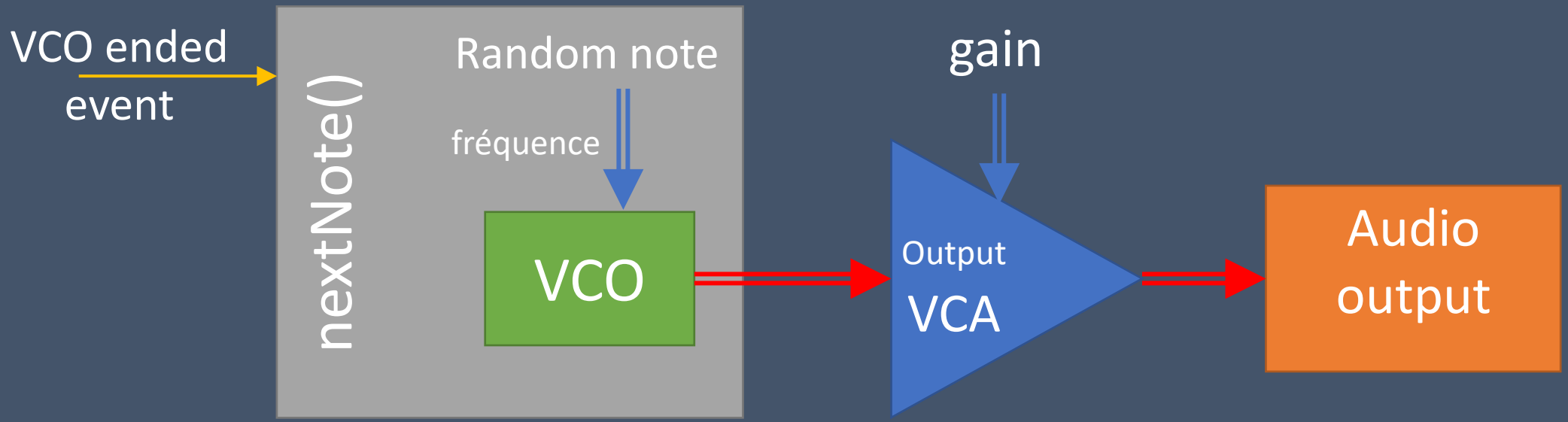
Définition de la gamme: ici une gamme majeure

Conversion du nombre random en note de musique

Apres un stop il faut recréer un nouvel Oscillateur

Demo 2 Random Notes





ADSR (enveloppe)

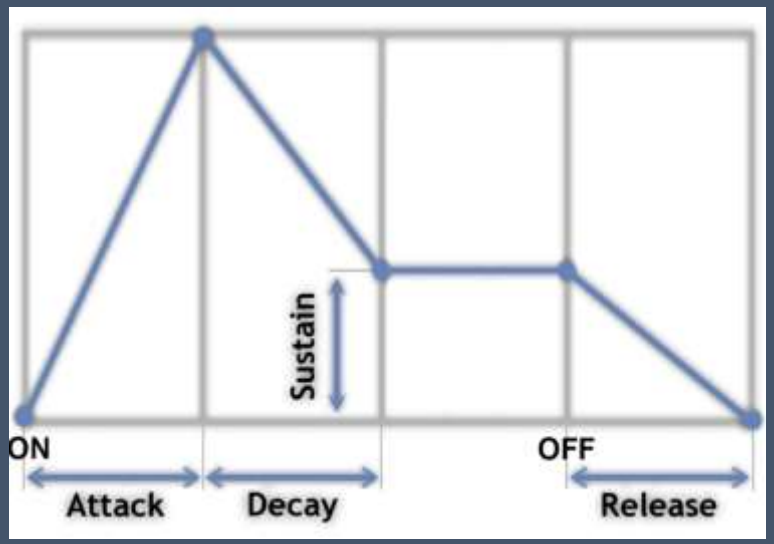
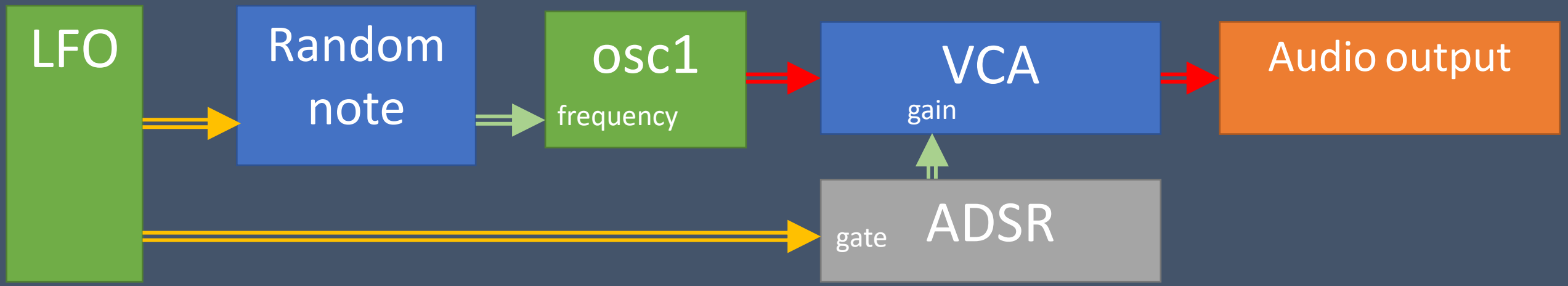


Image source: wikipedia



ADSR en JavaScript

```
const attack_time = duration * adsr_panel.attack.value / 100;
const decay_time = (duration - attack_time) * adsr_panel.decay.value / 100;
const sustain = adsr_panel.sustain.value / 100;
const release_time = (duration - attack_time - decay_time) * adsr_panel.release.value / 100;

adsr_vca.gain.linearRampToValueAtTime(1, now + attack_time);
adsr_vca.gain.linearRampToValueAtTime(sustain, now + attack_time + decay_time);
adsr_vca.gain.linearRampToValueAtTime(sustain, now + duration - release_time);
adsr_vca.gain.linearRampToValueAtTime(0, now + duration);
```

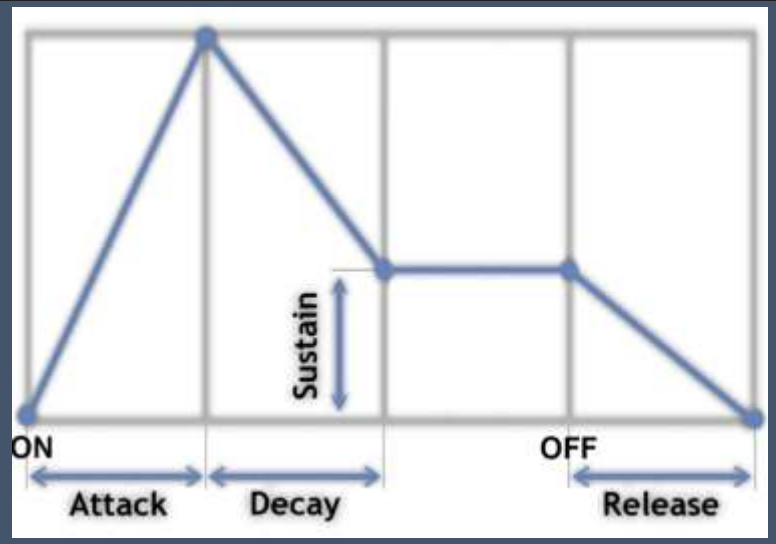
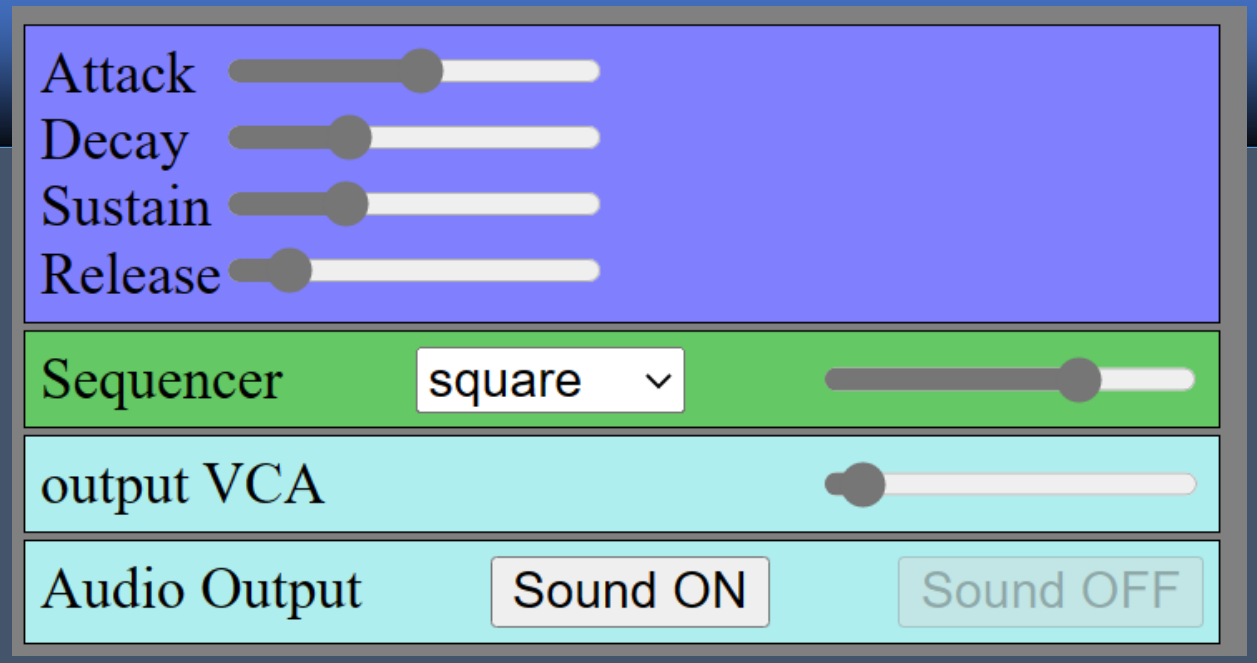
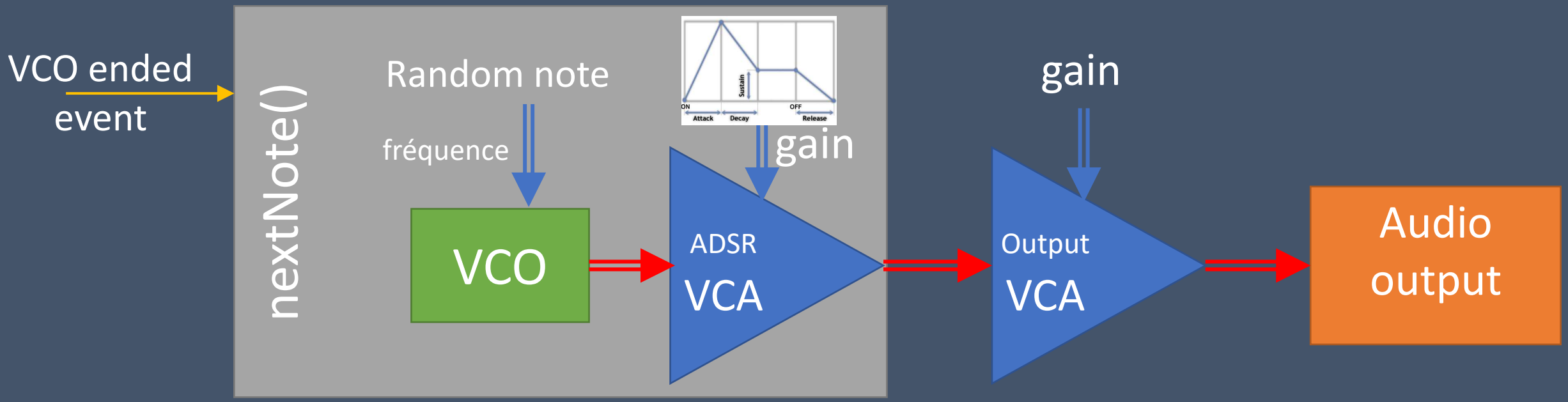


Image source: wikipedia

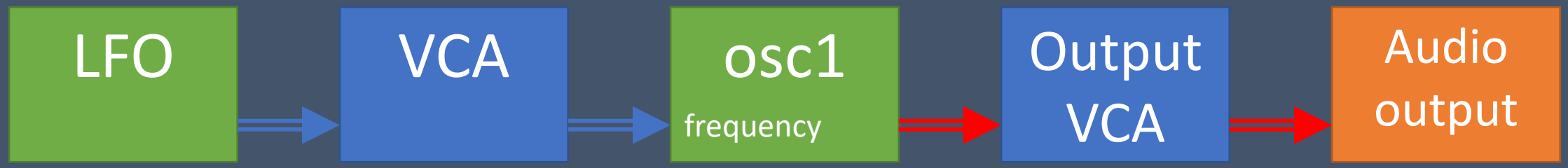
Utilisation de la méthode « LinearRampToValue » de l'audioParam « gain » pour tracer la courbe

Demo 3 ADSR

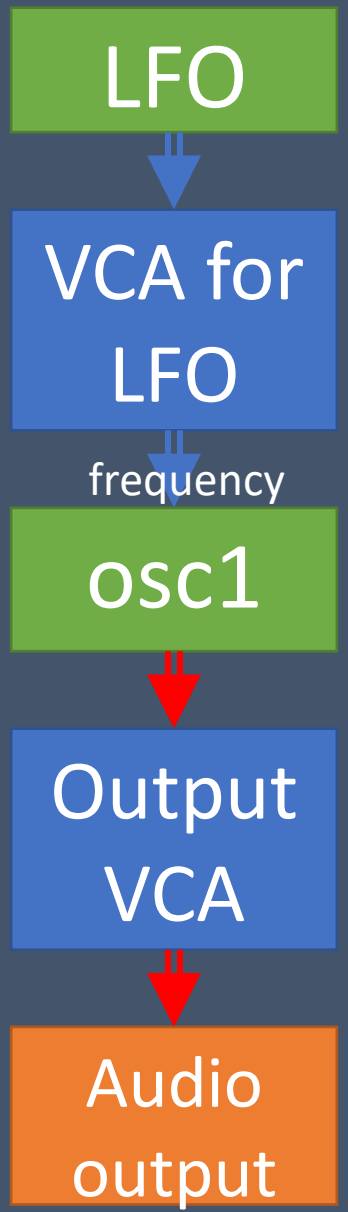

Attack
Decay
Sustain
Release
Sequencer
square
output VCA
Audio Output
Sound ON
Sound OFF


VCO ended
event
nextNote()
Random note
fréquence
VCO
Sustain
ON
OFF
Attack
Decay
Release
gain
ADSR
VCA
gain
Output
VCA
Audio
output

Le LFO contrôle un VCO



Connecter un node à un **audioParam**

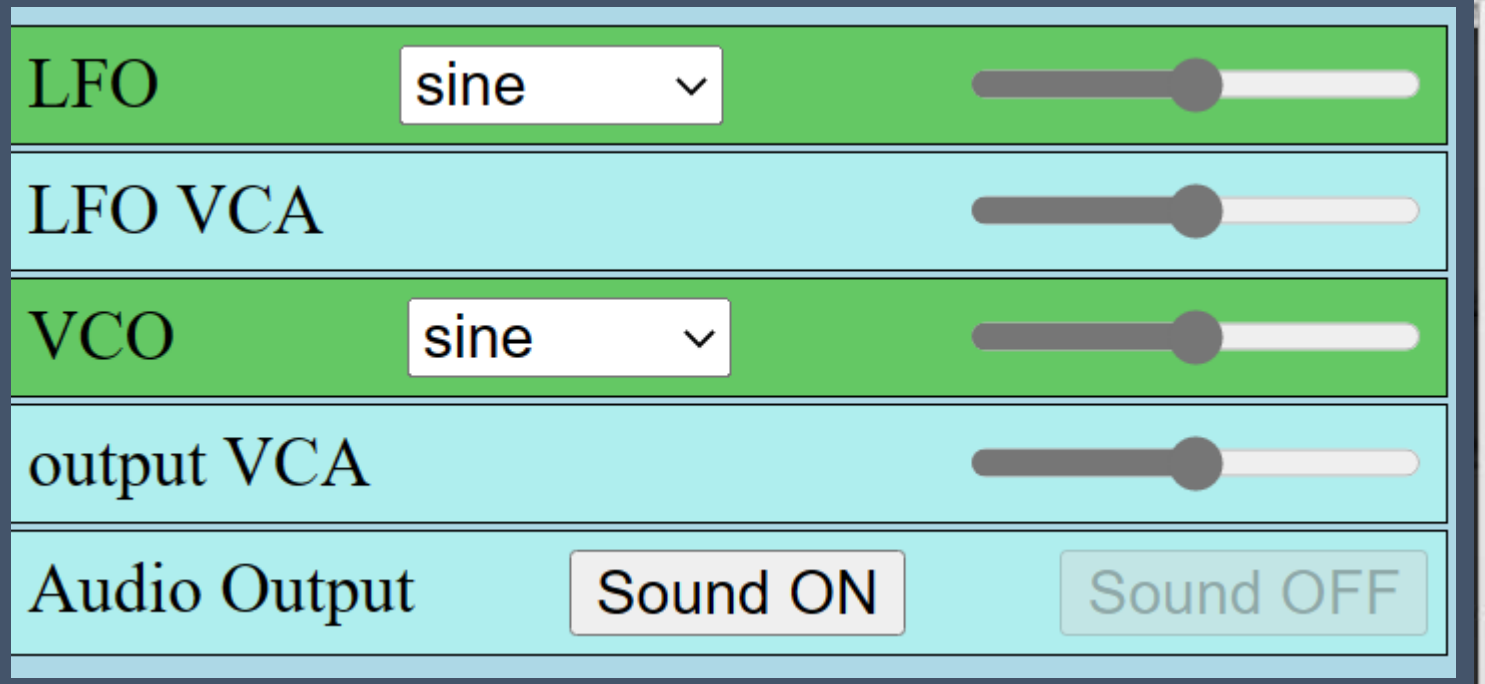



```
//create the LFO
const lfo = audio_ctx.createOscillator();
lfo.type = "sine";
lfo.frequency.value = 10;
lfo.start();

//create the LFO amplifier
const vca_for_lfo = audio_ctx.createGain();
vca_for_lfo.gain.value = 250;

// create the tone generator
const osc1 = audio_ctx.createOscillator();
osc1.type = 'sine';
osc1.start();

//====================== Connect the nodes ==========
lfo.connect(vca_for_lfo);
vca_for_lfo.connect(osc1.frequency);
osc1.connect(output_vca);
output_vca.connect(audio_output);
```

Valeur de gain élevée pour obtenir une fréquence convenable

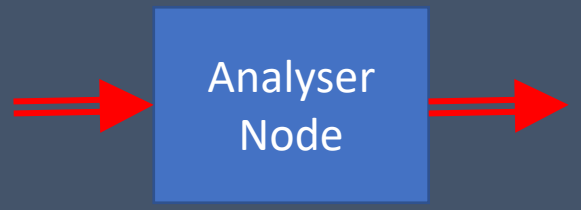
Le control Voltage est implémenté en connectant la sortie de l'audiNode à un audioParam

Demo4: VCO Contrôlé par LFO



L’oscilloscope

- Sert à afficher la forme d’onde
- Implémenté par un AnalyserNode
- La sortie est une copie de l’entrée



Méthodes:

AnalyserNode.getFloatTimeDomainData()

AnalyserNode.getByteTimeDomainData()

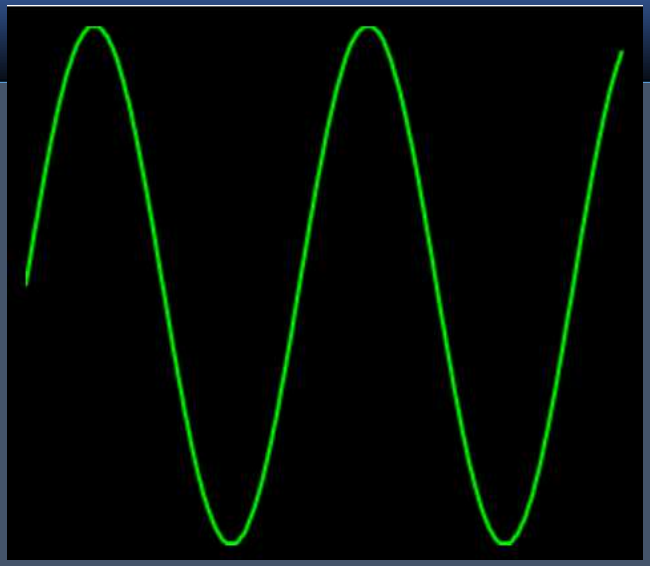
Attribut:

`AnalyserNode.fftSize` taille de l’array (values: 32, 64,…32768)

L’oscilloscope: le code

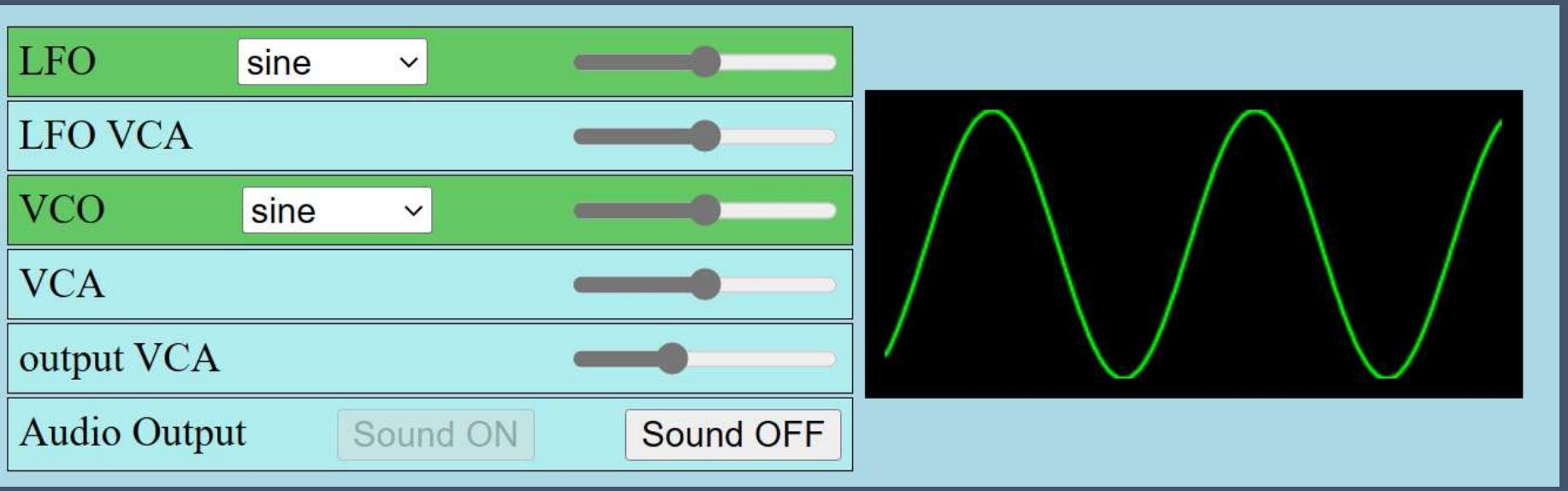
```
//create an analyser for oscilloscope
const analyser = audio_ctx.createAnalyser();

const canvas = document.getElementById("canvas_oscilloscope");
canvas.width = 300;
canvas.height = 130;
const canvasCtx = canvas.getContext("2d");
const bufferLength = 512;
analyser.fftSize = bufferLength;
```

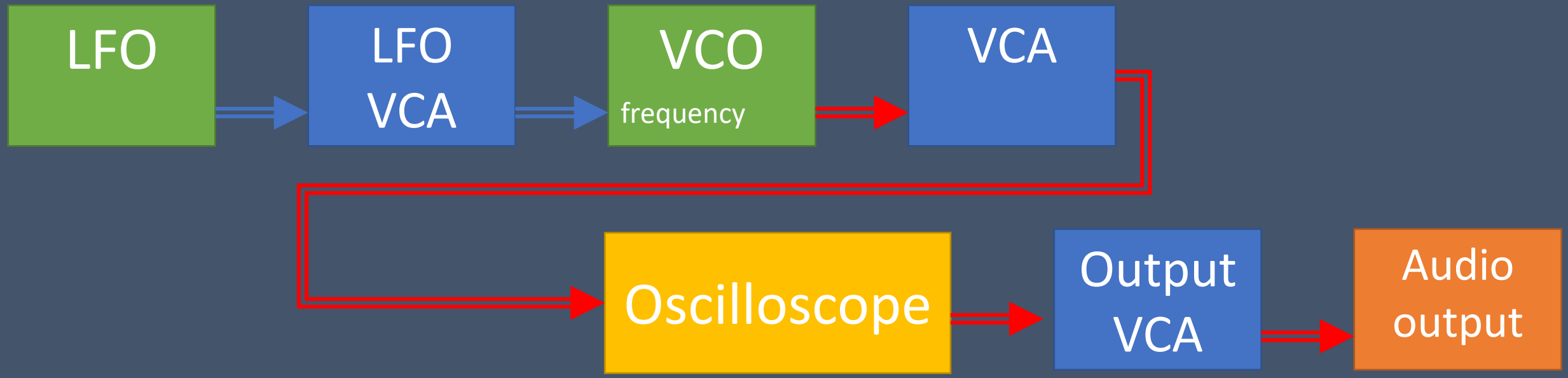
L’oscilloscope: le code (affichage)



```
function draw() {
    const dataArray = new Uint8Array(bufferLength);
    analyser.getByteTimeDomainData(dataArray);

    canvasCtx.clearRect(0, 0, canvas.width, canvas.height);
    canvasCtx.lineWidth = 2;
    canvasCtx.strokeStyle = "rgb(0, 255, 0)";

    const sliceWidth = ((canvas.width * 1.0) / bufferLength) * 2;
    let x = 0;
    canvasCtx.beginPath();
    for (let i = 0; i < bufferLength; i++) {
        const y = canvas.height - (((dataArray[i] / 128.0) * canvas.height) / 2);
        if (i === 0) { canvasCtx.moveTo(x, y); }
        else { canvasCtx.lineTo(x, y); }
        x += sliceWidth;
    }
    canvasCtx.stroke();

    requestAnimationFrame(draw);
}

draw();
```

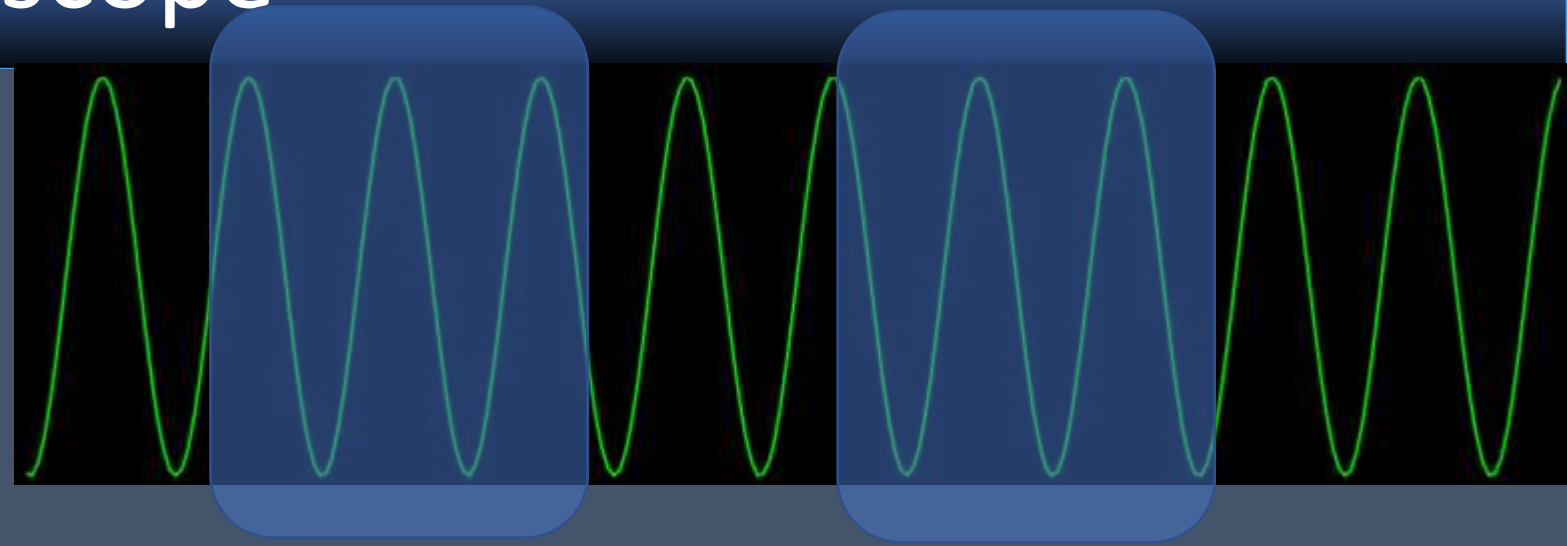
Demo4: Oscilloscope





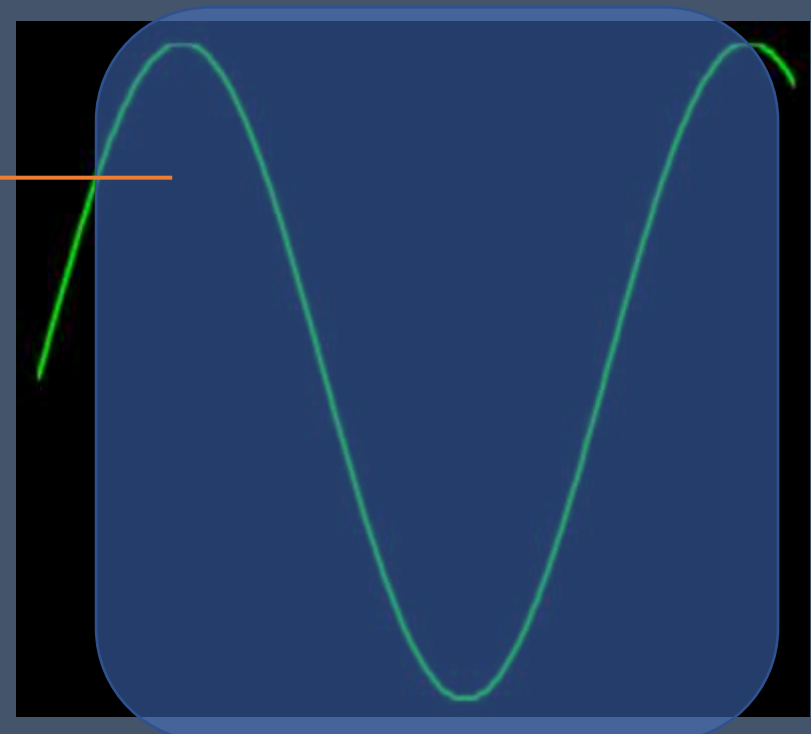
Le trigger de l'oscilloscope

Stabiliser l'image



Toujours commencer à dessiner
au même moment (hauteur / pente)

- Trigger level
- Trigger slope ou

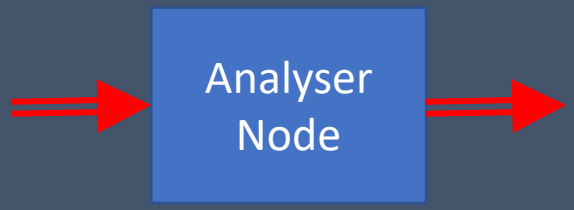


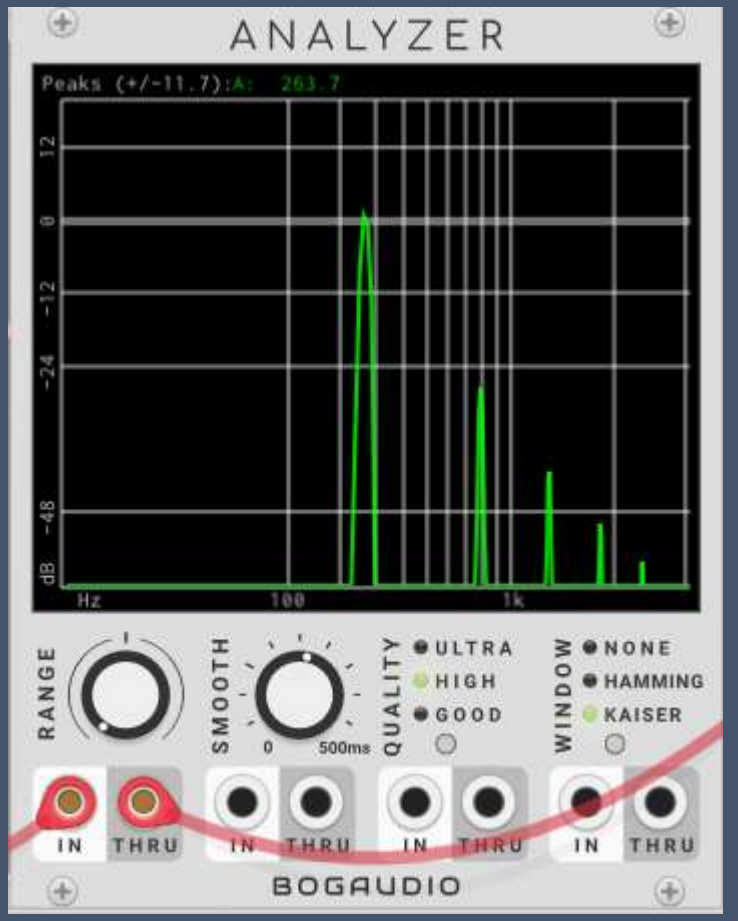
Implémentation du trigger de l'ocsilloscope

```
canvasCtx.beginPath();
let startindex = 0;
const trig_level = document.getElementById("slider_trigger_level").value;
while (
    startindex < bufferLength - 2 &&
    !(
        dataArray[startindex] > trig_level &&
        dataArray[startindex + 2] < trig_level
    )
) {
    startindex++;
}
if (startindex === bufferLength - 2) {
    //not triggered
    startindex = 0;
}
for (let i = 0; i < bufferLength / 2; i++) {
    const y = canvas.height - (((dataArray[i + startindex] / 128.0) * canvas.height) / 2);
```

Le spectre (FFT)

- Sert à afficher le contenu fréquentiel du signal
- Implémenté par un **AnalyserNode**





Méthodes:

AnalyserNode.getFloatFrequencyData()

AnalyserNode.getByteFrequencyData()

Attributs:

`AnalyserNode.fftSize` (values: 32,64,128,…32768)

`AnalyserNode.frequencyBinCount`= fftSize / 2 (taille du dataArray)

new_5_spectrum.vcv

Code du spectrum analyser

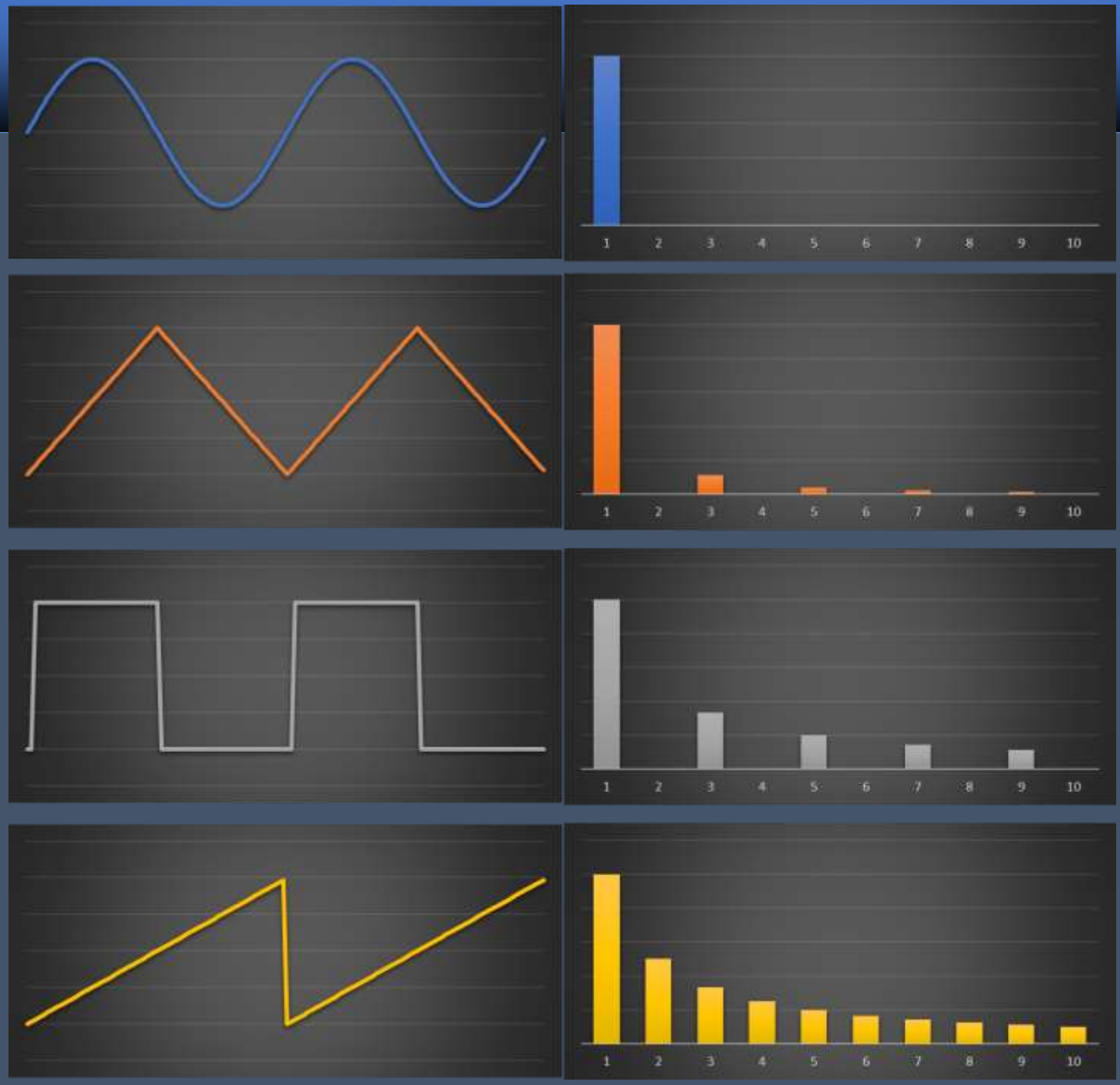
```
//create an analyser for spectrum
const analyser_spectrum = audio_ctx.createAnalyser();

const canvas_spectrum = document.getElementById("canvas_spectrum");
canvas_spectrum.width = 300;
canvas_spectrum.height = 130;
const canvas_spectrum_ctx = canvas_spectrum.getContext("2d");
analyser.fftSize = 512;
```

```
function draw_spectrum() {
    const dataArray = new Uint8Array(analyser_spectrum.frequencyBinCount);
    analyser_spectrum.getByteFrequencyData(dataArray);

    canvas_spectrum_ctx.clearRect(0, 0, canvas_spectrum.width, canvas_spectrum.height);
    canvas_spectrum_ctx.beginPath();
    let x = 0;
    canvas_spectrum_ctx.lineWidth = 2;
    canvas_spectrum_ctx.strokeStyle = "rgb(192, 128, 0)";
    for (let i = 0; i < analyser_spectrum.frequencyBinCount; i++) {
        if (i === 0) { canvas_spectrum_ctx.moveTo(x, canvas_spectrum.height - (dataArray[i] * canvas_spectrum.height / 256));
        else { canvas_spectrum_ctx.lineTo(x, canvas_spectrum.height - (dataArray[i] * canvas_spectrum.height / 256)); }
        x += (canvas_spectrum.width * 1.0) / analyser_spectrum.frequencyBinCount;
    }
    canvas_spectrum_ctx.stroke();

}
```

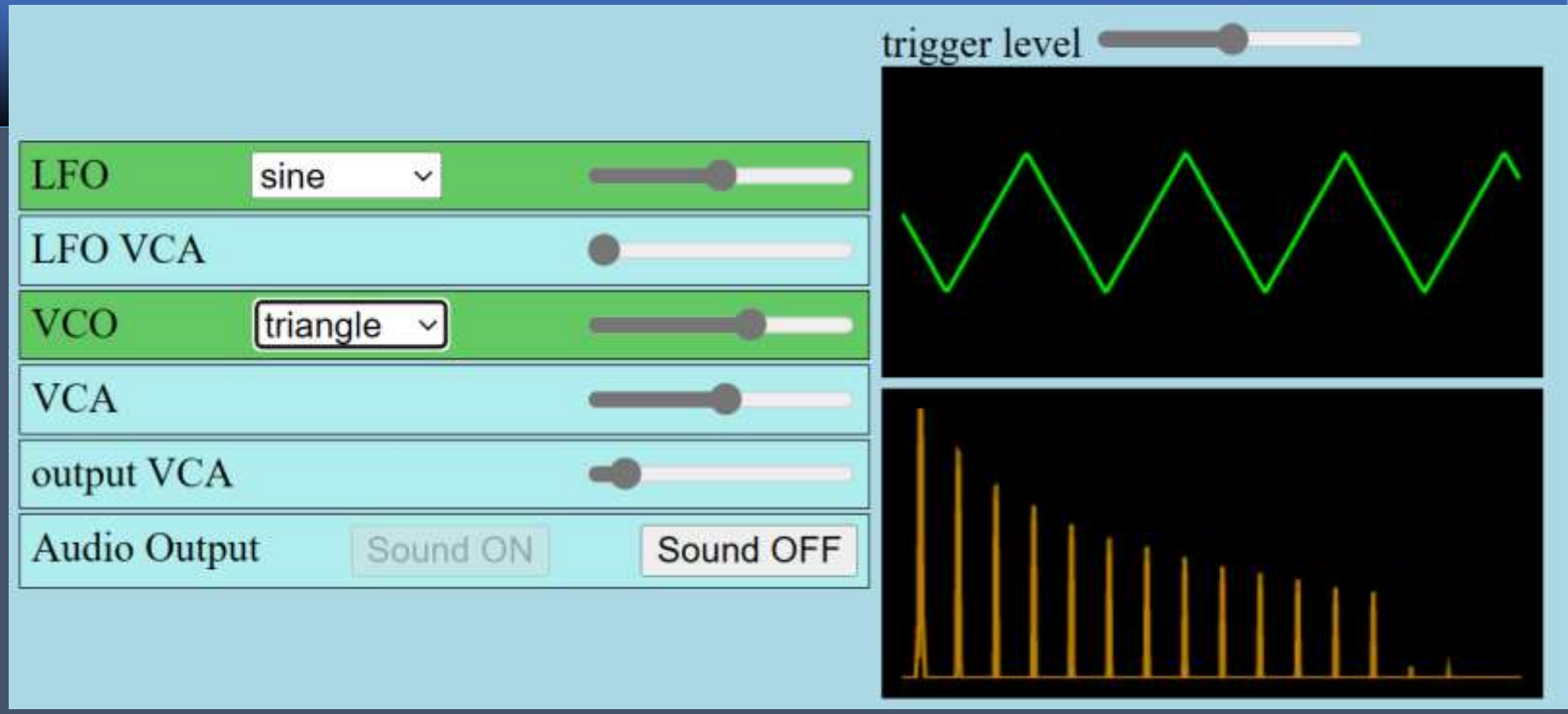
Similaire au code de l'oscilloscope

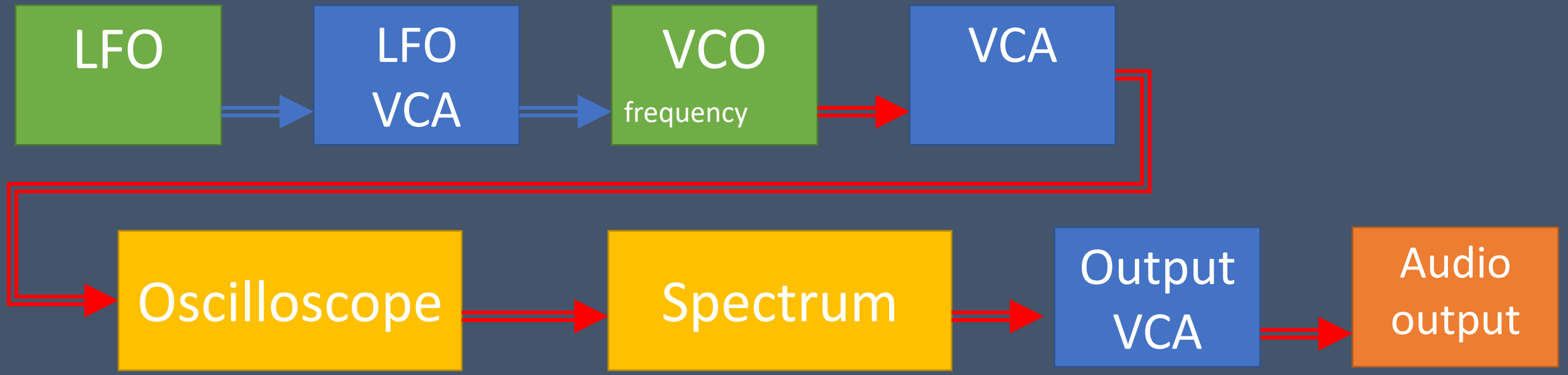
Pas de trigger

Les harmoniques



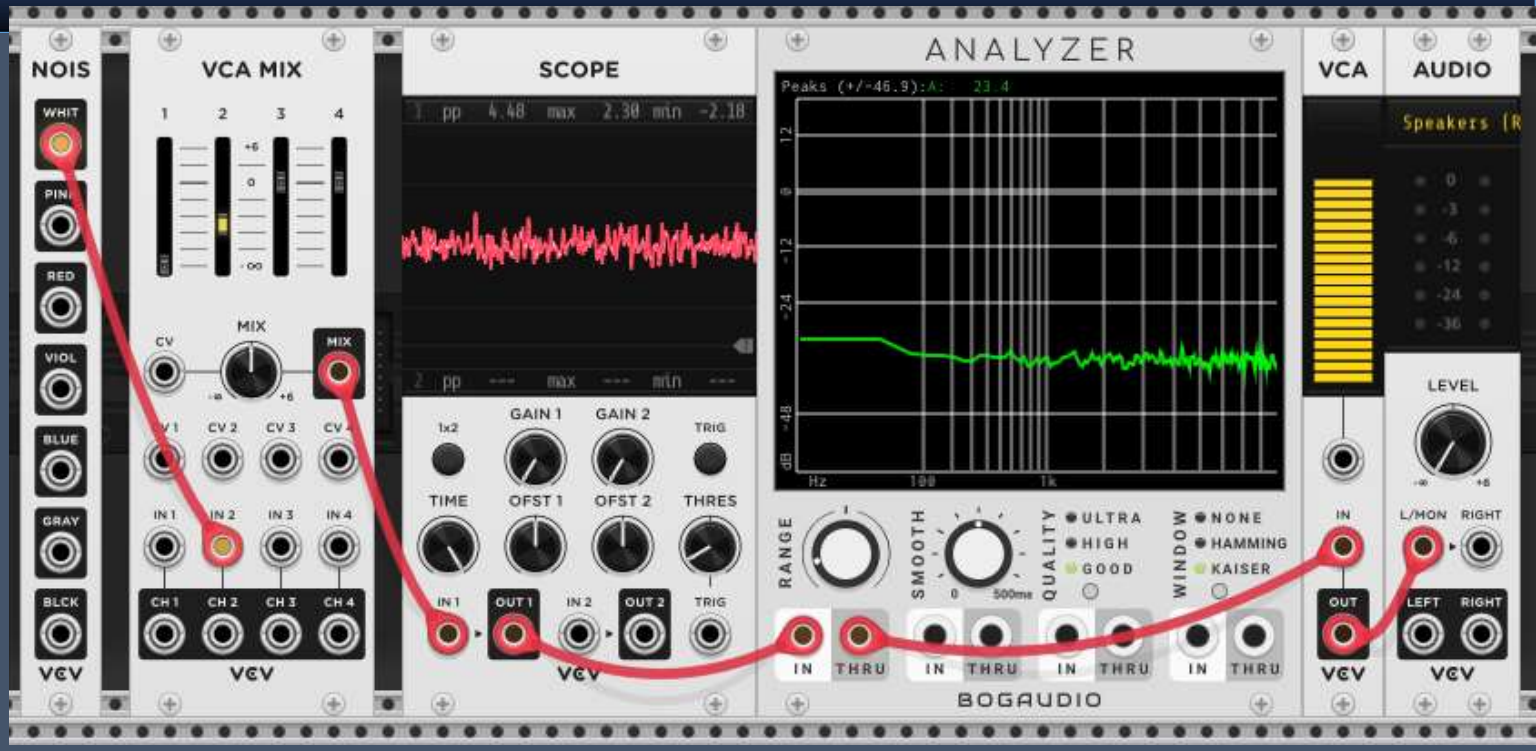
Demo 6 Spectrum





Générer du bruit blanc

- Le bruit blanc a un spectre plat



- Le signal = random numbers (distribution uniforme)

AudioBufferSourceNode

- L' **AudioBufferSourceNode** génère un signal audio à partir de samples stockés dans un **AudioBuffer**

Méthodes:

AudioBufferSourceNode.start()

Attributs:

AudioBufferSourceNode.buffer AudioBuffer

AudioBufferSourceNode.loop boolean

```
//create a buffer source
const noise_source = audio_ctx.createBufferSource();
noise_source.buffer = noise_buffer;
noise_source.loop = true;
noise_source.start();
```

AudioBuffer

L'**audioBuffer** contient les données qui seront générées par l'**AudioBufferSourceNode**

Création:

`AudioContext.CreateBuffer(nbrOfChannels,length,sampleRate)`

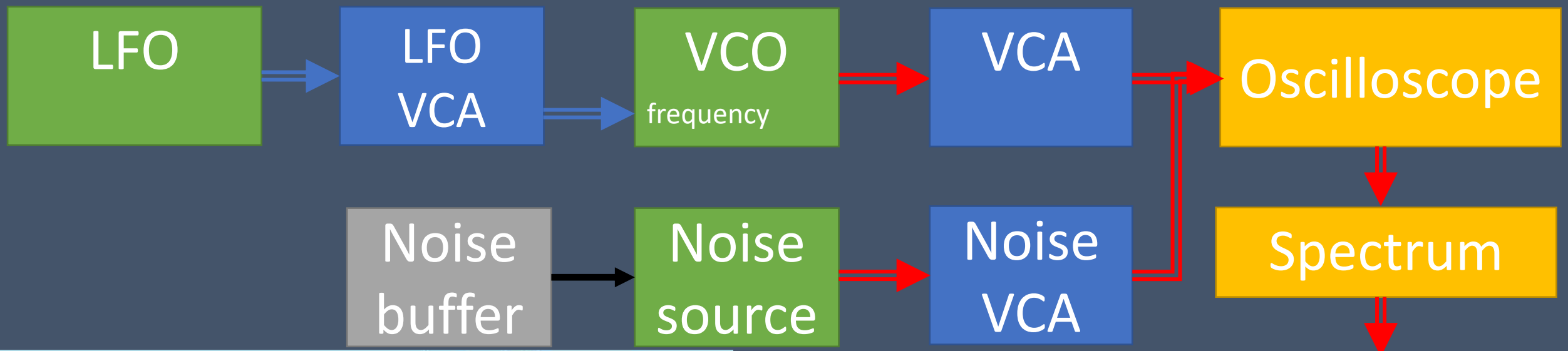
length = sampleRate*duration[s]

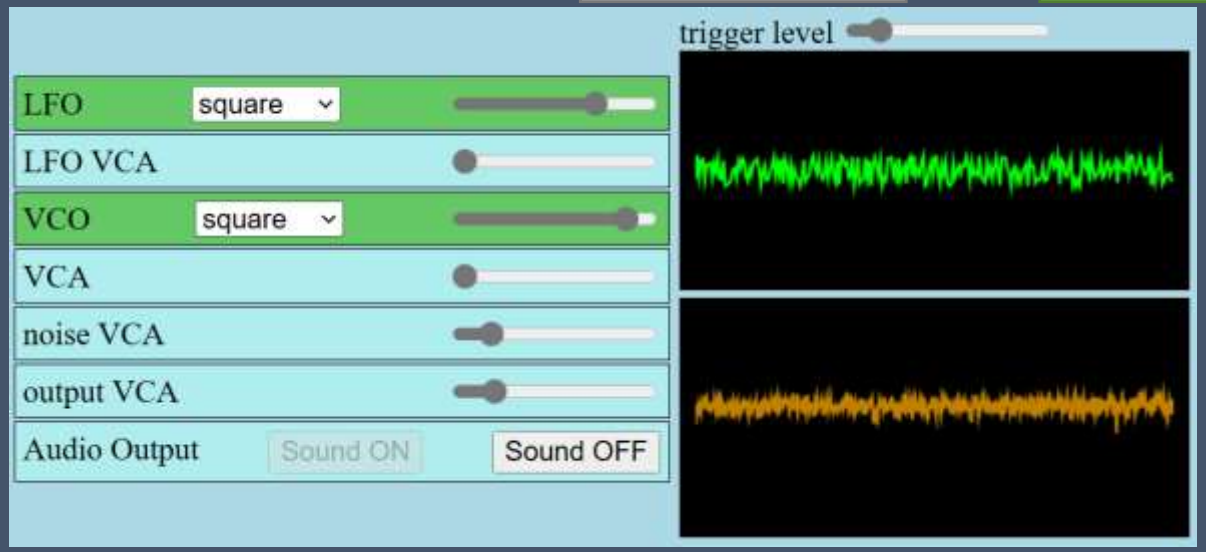
Méthodes:

`AudioBuffer.getChannelData(channel)` Float32Array

```
const noise_buffer = audio_ctx.createBuffer(1, audio_ctx.sampleRate * 3, audio_ctx.sampleRate);
const noiseData = noise_buffer.getChannelData(0);
for (let i = 0; i < noiseData.length; i++) {
    noiseData[i] = Math.random() * 2 - 1;
}
```

Demo 8 Noise


LFO
LFO VCA
VCO
frequency
VCA
Oscilloscope
Noise buffer
Noise source
Noise VCA
Spectrum
Output VCA
Audio output


trigger level
LFO
square
LFO VCA
VCO
square
VCA
noise VCA
output VCA
Audio Output
Sound ON
Sound OFF

Manipuler les sons: les filtres

Les filtres atténuent certaines parties du Spectre

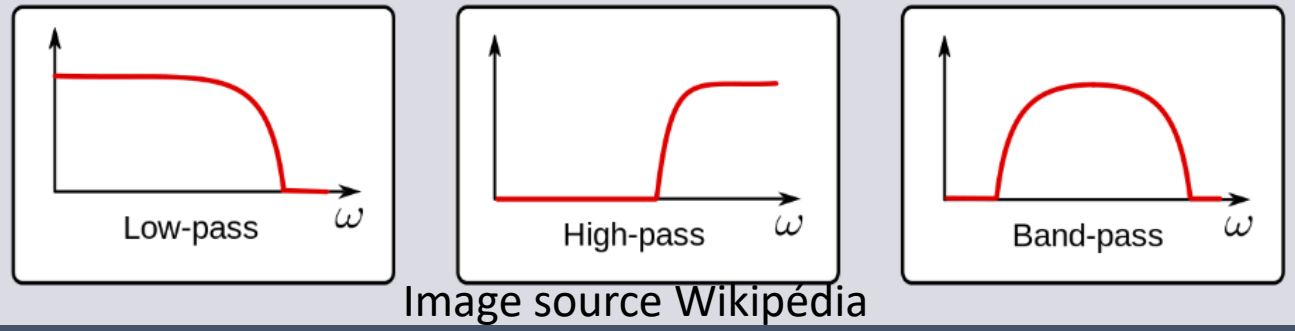
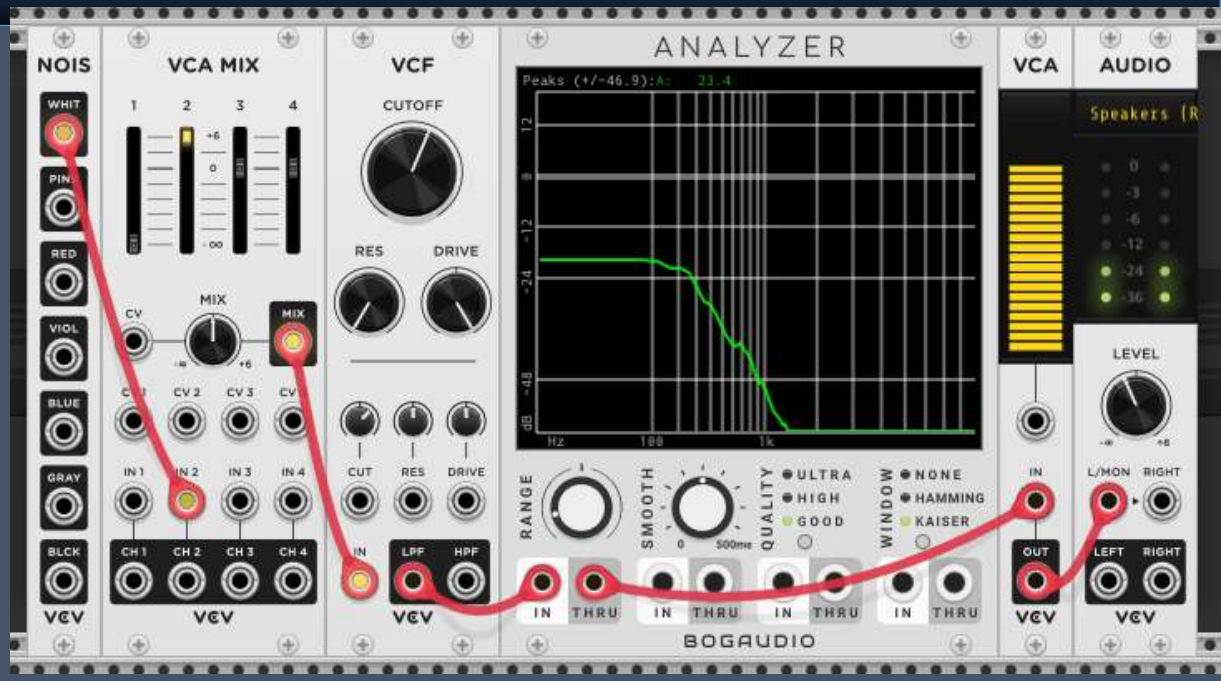


Image source Wikipédia



la synthèse [soustractive] provient du filtrage fréquentiel qui va atténuer une partie du spectre du son, et donc modifier son timbre en ne conservant qu'une partie de ses harmoniques. (source: Wikipédia)

BiquadFilterNode

- Attributs

  `BiquadFilterNode.type` lowpass, highpass,bandpass…

- AudioParams

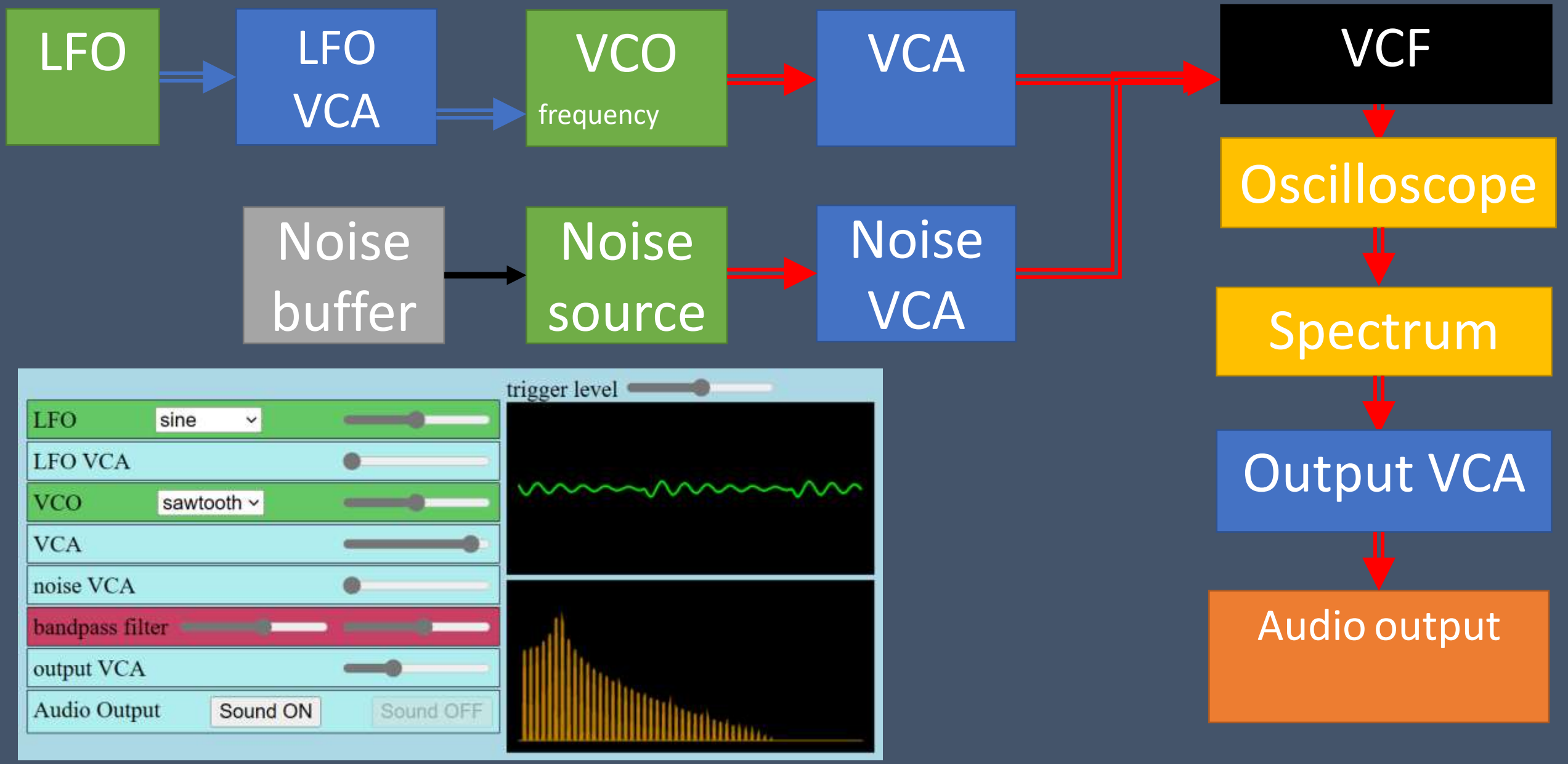
  `BiquadFilterNode.Q` largeur du bandpass et pic du lowpass et du highpass

  `BiquadFilterNode.frequency` fréquence du filtre

Biquad Filter node code

```
const filter = audio_ctx.createBiquadFilter();
filter.type = "bandpass";
const filter_panel = create_filter_panel("bandpass filter");
filter_panel.frequency.addEventListener("input", (event) => {
    filter.frequency.setValueAtTime(event.target.value * 50, audio_ctx.currentTime);
});
filter_panel.q.addEventListener("input", (event) => {
    filter.Q.setValueAtTime(event.target.value / 5, audio_ctx.currentTime);
});
```

Demo 9: Filtres



Conclusion

- La Web Audio API
  - Disponible dans tous les web browsers modernes
  - Permet de simuler très précisément le comportement d'un synthétiseur modulaire
  - Facile à mettre en oeuvre

- Application
  - Jeux (création des sons en fonction de la situation du jeu)
  - Musique
  - Education
  - Générateur de signaux (basse fréquence) pour un labo d'électronique