Sample Code

Building a signal generator

Generate audio signals using an audio source node and a custom render callback.

Download

iOS 17.0+ | iPadOS 17.0+ | macOS 14.0+ | Xcode 16.4+

Overview

This sample code project shows you how to use an audio source node and a custom render callback to generate classic waveforms.

> **Note**
>
> This sample code project is associated with WWDC19 session 510: What's New in AVAudioEngine.

A class called `SignalGeneratorKernel`, written in C++ to ensure real-time safety, provides the digital signal processing (DSP) logic. The project also includes an Objective-C wrapper class called `SignalGenerator` to act as an intermediary.

> **Important**
>
> To ensure glitch-free performance, audio processing must occur in a real-time safe context. Don't allocate memory, perform file I/O, take locks, or interact with the Swift or Objective-C runtimes when rendering audio.

You can change the waveform, frequency, and amplitude of the signal generator in real time. Because changing the amplitude and frequency of the signal generator can produce audible artifacts, the sample project uses the included `ParameterRamp` class to ramp these parameters.

Synthesize classic waveforms

The classic waveforms the signal generator kernel produces are sine, sawtooth down, square, triangle, and white noise. Digital synthesis of classic waveforms that have discontinuities can produce aliasing. This is particularly the case for sawtooth and square waveforms, which have jump discontinuities.

Tip

Frequency is the derivative of phase. When this derivative is undefined, such as due to a discontinuity, aliasing can occur because the frequencies that can be represented in a digital signal are limited by the sample rate.

To mitigate aliasing, the sample generates sawtooth, square, and triangle waveforms after their Fourier series. Although not efficient, this approach ensures the waveforms are band-limited, and produces the maximum number of harmonics with frequencies bound by the Nyquist frequency (half the sample rate).

```
void setSampleRate(float inSampleRate) {
    // Store the sample rate.
    sampleRate = inSampleRate;
    // Update the maximum number of harmonics by taking the floor of the Nyquist fre
    numHarmonics = int(0.5f * sampleRate / frequency);
    // Set the phase increment ramp length to 100 milliseconds.
    phaseIncrement.setRampLength(0.1f * sampleRate);
    // Set the raw amplitude ramp length to 100 milliseconds.
    rawAmplitude.setRampLength(0.1f * sampleRate);
}
```

The Fourier series for a sawtooth down waveform is a sum of harmonic partials that decays proportionally to the partial’s frequency.

```
inline float additiveSawtooth(float phase, int harmonics) {
    float sample = 0;

    for (int i = 1; i <= harmonics; ++i)
        sample += sin(i * phase) / i;
    return (2.0f / M_PI) * sample;
}
```

Square and triangle waves are sums of odd partials. The square decays proportionally to the the partial's frequency, and the triangle decays proportionally to the square of the partial's frequency.

```
inline float additiveSquare(float phase, int harmonics) {
    float sample = 0;

    for (int i = 1; i <= harmonics; i += 2)
        sample += sin(i * phase) / i;

    return (4.0f / M_PI) * sample;
}

inline float additiveTriangle(float phase, int harmonics) {
    float sample = 0;

    for (int i = 1; i <= harmonics; i += 2)
        sample += powf(-1, (i - 1) / 2) * sin(i * phase) / (i * i);

    return (8.0f / (M_PI * M_PI)) * sample;
}
```

Define a custom render callback

The `AudioManager` class bridges an instance of `SignalGenerator` to an AVAudioEngine. When initialized, `AudioManager` constructs an AVAudioSourceNode object using the `renderCallback` property of the signal generator class. After creating an audio source node, `AudioManager` attaches the node to the AVAudioEngine, and connects it to the main mixer node.

```
init() {
    let srcNode = AVAudioSourceNode(renderBlock: signalGenerator.renderBlock)
    let mainMixer = engine.mainMixerNode
    ...
    engine.attach(srcNode)
    engine.connect(srcNode, to: mainMixer, format: inputFormat)
    engine.connect(mainMixer, to: output, format: outputFormat)
    mainMixer.outputVolume = 0.5
}
```

`SignalGenerator` provides a render block in much the same way an instance of AUAudioUnit provides an internalRenderBlock. The basic difference is that `SignalGenerator` provides a render block of type AVAudioSourceNodeRenderBlock, and AUAudioUnit provides a render block of type AUInternalRenderBlock.

```
@interface SignalGenerator : NSObject

// The block this class provides to implement rendering. Similar to `AUInternalRende
@property (nonatomic, readonly) AVAudioSourceNodeRenderBlock renderBlock;

...

@end
```

Connect the user interface to the signal generator

The user interface is a View that observes a property of type `AudioManager`. The audio manager class, in turn, implements the Observable protocol, and is responsible for maintaining the state of the user interface.

The audio manager exposes properties that control the waveform, frequency, and amplitude of the signal generator, and provides methods to start and stop the AVAudioEngine.

```
var waveform: Waveform = .sine {
    willSet {
        signalGenerator.setWaveform(newValue)
    }
}

var frequency: Float = 440.0 {
    willSet {
        signalGenerator.setFrequency(newValue)
    }
}

var amplitude: Float = -12.0 {
    willSet {
        signalGenerator.setAmplitude(newValue)
    }
}
```

See Also

Rendering

{} Performing offline audio processing

Add offline audio processing features to your app by enabling offline manual rendering mode.

`class AVAudioSourceNode`

An object that supplies audio data.

`class AVAudioSinkNode`

An object that receives audio data.