Complex Rhythmic Structures with Beet.js

Ehsan Ziya
London, UK
ehsan.ziya@gmail.com

ABSTRACT

This paper describes `Beet.js`, a new Javascript library for creating complex and layered rhythmical structures in the browser using the Web Audio API.

Link to source: github.com/zya/beet.js

Link to demo page: zya.github.io/beet.js

Categories and Subject Descriptors

H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

Web Audio API, Polyrhythm, Euclidean Rhythm, JavaScript, Generative Music

1. INTRODUCTION

The Web Audio API[1] has introduced a wide range of new opportunities for creating new musical experiences and more importantly, delivering those experiences to a wide number of audiences through the browser. Using the real-time client-side audio rendering and synthesis capabilities provided by the browser, music can now be delivered to browsers not as traditional static audio files, but as set of assets, instructions and rules that describe a dynamic musical system.

One of the aspects of creating dynamic musical systems is rhythm. `Beet.js` [2] was written to provide a tool-set for creating complex rhythmical structures in the browser using the Web Audio API. The library uses a multi-layered approach to sequencing where each layer operates independently and can follow a different pattern, time signature and speed. In addition, built-in methods for generating evenly distributed patterns (Euclidean patterns)[3] and a simple API for layering allow the users to generate complex rhythmic structures that can be used for creating real-time generative and/or interactive musical experiences for the web. The library was written with focus on providing a thin abstraction layer over sequencing and event scheduling as well as pattern generation, but does not offer any sound generation features. This will allow users to use `Beet.js` with existing frameworks and libraries easily. In the next section we will examine each of the features more in detail.

Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA.

2. TECHNICAL OVERVIEW

2.1 Installation and Initialisation

`Beet.js` is Browserify [4] compatible and can be installed using NPM [5] using the command below. To initialise an instance of `Beet.js`, follow the code example below.

```
$ npm install beet.js

var Beet = require('beet.js');
var context = new AudioContext();
var beet = new Beet({
  context: context,
  tempo: 120 // in bpm
});
```

Listing 1: Beet.js Installation and Initialisation

2.2 Patterns

One of the main features of the library is pattern generation based on Bjorklund's [6] equal distribution algorithm. The algorithm solves the general problem of distributing `n` pulses over `m` timing slots in the most even way possible, even though `n` may not necessarily be an even divisor of `m`. For example, it equally distributes 5 pulses over 9 timing slots and generates the binary sequence: `101110111`. In Figure 1, you can see the visualisation of the pattern `5,9` on the left and `9,13` on the right.

The algorithm was initially developed to be used with spallation neutron source (SNS) [7] accelerators in nuclear physics but Godfried Toussaint proved that the algorithm can be used to generate a large family of rhythms found in sub-Saharan African music and world-music in general [3].

`Beet.js` exposes a `pattern` method that returns a `Pattern` object based on a number of pulses to be distributed over a number of slots. The pattern object can later be used with `Layer` objects to schedule events. This method uses a recursive implementation of the Bjorklund algorithm that can also be found as a separate library [8].

```
beet.pattern(5, 8); // returns 10101101 — 5 over 8
beet.pattern(2, 5); // returns 10100 — 2 over 5
beet.pattern(4); // returns 1111 — 4 over 4
```

Listing 2: Even Distribution Patterns based on the Bjorklund algorithm [6]

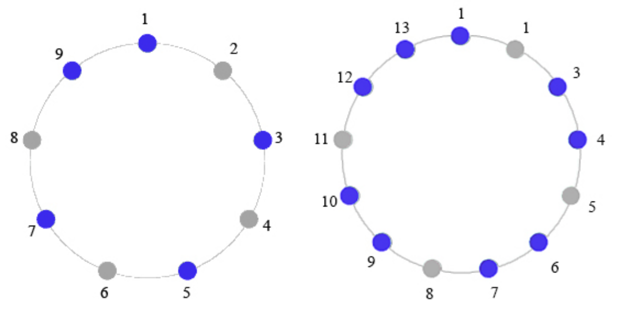

Figure 1: (left) Euclidean rhythm (5, 9), (right) Euclidean rhythm (9, 13)

There is also a method available on the `pattern` objects to shift the sequence by any given number. This can be used to create variations on each sequence.

```
var pattern = beet.pattern(5, 9); // 101110111
pattern.shift(1); // 011101111
var pattern2 = beet.pattern(1, 4); // 1000
pattern2.shift(3); // 0001
```

Listing 3: Pattern Shift

2.3 Layers

Layers are the main building blocks of `Beet.js`. Each `Layer` object consists of a `Pattern` object and two callbacks: one for `pulses` and one for the rest of the slots. For example for the pattern `1010`, the first callback will be invoked on `1`s and the second will be invoked on `0`s. This feature allows for more flexibility when creating rhythmic structures.

An instance of `Layer` object can be created using the `beet.layer` method which takes a pattern and two callbacks of which the second is optional. After creation, each layer can be added or removed using `beet.add` and `beet.remove` methods.

```
var pattern = beet.pattern(1, 4); // 1000
var layer = beet.layer(pattern, cbForPulses,
    cbForRest);
beet.add(layer);
```

Listing 4: Layers and Callbacks

As mentioned before, one of the main features of `Beet.js` is that each layer operates independently from other layers. That is: each layer uses a separate worker thread for scheduling events therefore can follow a different tempo than other layers. This makes it possible to create evolving rhythmic sequences using different layers with different lengths and speeds.

```
// create first layer with tempo of 100 bpm
var pattern1 = beet.pattern(1, 4);
var layer1 = beet.layer(pattern, cb1);
layer1.tempo = 100;
beet.add(layer1);
// create a second layer with tempo of 80 bpm
var pattern2 = beet.pattern(5, 9);
var layer2 = beet.layer(pattern2, cb2);
layer2.tempo = 80;
beet.add(layer2);
beet.start(); // start beet
```

Listing 5: Independent Layers with Differenet Speeds

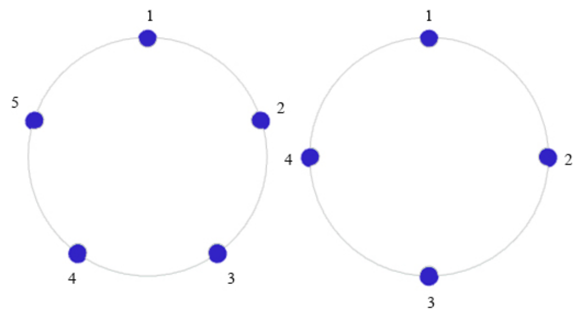

Figure 2: Polyrhythm - 5s (left) over 4s (right)

Using this feature, it is also possible to create polyrhythms [9]. Figure 2 is a visualisation of a simple polyrhythmic sequence (Listing 6) created using two layers with different step counts.

```
var fours = beet.pattern(4); // 1111
var firstLayer = beet.layer(fours, foursCB);
var fives = beet.pattern(5); // 11111
var secondLayer = beet.layer(fives, fivesCB);
beet.add(firstLayer);
beet.add(secondLayer);
beet.start();
```

Listing 6: Polyrhythm - 5 over 4

2.4 Callbacks

As mentioned before, each `Layer` can have two callbacks. The callback format is as shown on Listing 7. Each callback will be called with 3 parameters. `time` is the audio time that can be used to schedule audio events. `step` is the current step number for each callback which can be used to determine application status, such as note changes in the sequence. Finally, `timeFromScheduled` is a value in seconds that can be used to schedule JS real-time events such as animations synced to the audio events (Listing 7). `timeFromScheduled` is calculated for each step by subtracting `time` from `context.currentTime` [10].

These three parameters, in addition to the option to have callbacks for `on` and `off` steps, provide a flexible tool-set for creating complex rhythmic patterns.

```
function callback(time,step,timeFromScheduled) {
  // schedule audio events using time
  // use step number to determine application
     state
  setTimeout(function(){
    // trigger some js event such as animation
    // will be synced to the audio
  }, timeFromScheduled * 1000);
}
```

Listing 7: Callback Format

3. CONCLUSIONS AND FUTURE WORK

The presented library provides a minimal and specialised tool-set for musicians and developers to create complex and dynamic rhythmic structures in the browser. The thin layer of abstraction over Web Audio API that takes care of scheduling events using a simple API and will make integration with other libraries easily possible. The library makes use of Chris Wilson's [11] audio scheduling method and uses a Web Worker [12] for each layer. This results in rock-solid timing and prevents the audio timing to be interupted by the page visibility.

For future work, the author is working on a draft for a new feature set for `Layer` functionality whereby the user can chain layers or group layers in a way that they will behave as one. This will make it easier to handle larger amounts of layers. One of the other features that are in the road-map for `Beet.js` is swing [13] and the ability to add a human feel to each layer. Future technical work may also include further refinement of the API by exposing more adjustable scheduling parameters such as look-ahead time.

`Beet.js` is available on Github and NPM under a MIT license [2].

4. REFERENCES

[1] Web audio api. http://webaudio.github.io/web-audio-api/. Accessed: 2015-10-10.

[2] Beet.js. http://github.com/zya/beet.js. Accessed: 2015-10-07.

[3] Godfried Toussaint. The euclidean algorithm generates traditional musical rhythms. http://ehess.modelisationsavoirs.fr/atiam/biblio/Toussaint-BRIDGES2005.pdf, August 2005.

[4] Browserify. http://browserify.org/. Accessed: 2015-10-07.

[5] Node package manager. http://npm.org. Accessed: 2015-10-07.

[6] E. Bjorklund. The theory of rep-rate pattern generation in the sns timing system. https://ics-web.sns.ornl.gov/timing/Rep-Rate%20Tech%20Note.pdf. Accessed: 2015-10-07.

[7] Spallation neutron source. https://en.wikipedia.org/wiki/Spallation_Neutron_Source. Accessed: 2015-10-07.

[8] Ehsan Ziya. Bjorklund javascript library. https://github.com/zya/bjorklund. Accessed: 2015-10-07.

[9] Polyrhythms. https://en.wikipedia.org/wiki/Polyrhythm. Accessed: 2015-10-07.

[10] Audiocontext.currenttime. https://developer.mozilla.org/en-US/docs/Web/API/AudioContext/currentTime. Accessed: 2014-10-08.

[11] A tale of two clocks - scheduling web audio with precision. http://goo.gl/gJuBue. Accessed: 2014-10-08.

[12] Html5 web workers. http://www.w3schools.com/html/html5_webworkers.asp. Accessed: 2014-10-08.

[13] Swing (jazz performance style). https://en.wikipedia.org/wiki/Swing_(jazz_performance_style). Accessed: 2015-10-07.