Strudel: Algorithmic Patterns for the Web

Felix Roos
Lembach, France
flix91@gmail.com

Alex McLean
Then Try This
Sheffield/Penryn, UK
alex@slab.org

1. INTRODUCTION

This paper introduces StrudelCycles (generally known as just Strudel, including in the following), an alternative implementation of the TidalCycles live coding system, using the JavaScript programming language. It is an attempt to make live coding more accessible through creating a system that runs entirely in the browser, while opening Tidals approach to algorithmic patterns (Algorithmic Pattern 2020) up to modern audio/visual web technologies. The Strudel REPL is a live code editor dedicated to manipulating strudel patterns while they play, with builtin visual feedback. While Strudel is written in JavaScript, the API is optimized for simplicity and readability by applying code transformations on the syntax tree level, allowing language operations that would otherwise be impossible. The application supports multiple ways to output sound, including Tone.js, Web Audio nodes, OSC messages and WebMIDI. The project is split into multiple packages, allowing granular reuse in other applications. Apart from TidalCycles, it draws inspiration from prior projects like TidalVortex (TidalVortex Zero 2022), Gibber (Charlie and Joann 2012), Estuary (Ogborn and Beverley 2017) and Feedforward (Feedforward 2020).

2. PORTING FROM HASKELL

The original TidalCycles (generally known as just Tidal) is implemented as a domain specific language (DSL), embedded in the Haskell pure functional programming language, taking advantage of Haskells terse syntax and advanced, strong type system. Javascript on the other hand, is a multi-paradigm programming language, with a dynamic type system. Because Tidal leans heavily on many of Haskells more unique features, it was not clear whether it could meaningfully be ported to a multi-paradigm scripting language. However, this already proved to be the case with an earlier port to Python [TidalVortex; TidalVortex Zero (2022)], and we successfully implemented Tidals pure functional representation of patterns in Strudel, including partial application, and functor, applicative and monad structures. Over the past few months since the project started in January 2022, a large part of Tidals functionality has already been ported, including its mini-notation for polymetric sequences, and a large part of its library of pattern manipulations. The result is a terse and highly composable system, where just about everything is a pattern, that may be transformed and combined with other patterns in a myriad of ways.

Web Audio Conference WAC-2022, July 6–8, 2022, Cannes, France.

3. REPRESENTING PATTERNS

The essence of Tidal are Patterns. Patterns are abstract entities that represent flows of time as functions, by adapting a technique called pure functional reactive programming. Taking a time span as its input, a Pattern can output a set of events that happen within that time span. It depends on the structure of the Pattern how the events are located in time. From now on, this process of generating events from a time span will be called **querying**. Example:

```
const pattern = sequence(c3, [e3, g3]);
const events = pattern.query(0, 1);
console.log(events.map(e => e.show()))
```

In this example, we create a pattern using the `sequence` function and **query** it for the timespan from `0` to `1`. Those numbers represent units of time called **cycles**. The length of one cycle depends on the tempo, which defaults to one cycle per second. The resulting events are:

```
[{ value: 'c3', begin: 0, end: 1/2 },
{ value: 'e3', begin: 1/2, end: 3/4 },
{ value: 'g3', begin: 3/4, end: 1 }]
```

Each event has a value, a begin time and an end time, where time is represented as a fraction. In the above case, the events are placed in sequential order, where c3 takes the first half, and e3 and g3 together take the second half. This temporal placement is the result of the `sequence` function, which divides its arguments equally over one cycle. If an argument is an array, the same rule applies to that part of the cycle. In the example, e3 and g3 are divided equally over the second half of the whole cycle.

In the REPL, the user only has to type in the pattern itself, the querying will be handled by the scheduler. The scheduler will repeatedly query the pattern for events, which then will be used for playback.

4. MAKING PATTERNS

In practice, the end-user live coder will not deal with constructing patterns directly, but will rather build patterns using Strudels extensive combinator library to create, combine and transform patterns.

The live coder may use the `sequence` function as already seen above, or more often the mini-notation for even terser notation of rhythmic sequences. Such sequences are often treated only a starting point for manipulation, where they then are undergo pattern transformations such as repetition, symmetry, interference or randomisation, potentially at multiple timescales. Because Strudel patterns are represented as pure functions of time rather than as data structures, very long and complex generative results can be represented and manipulated without having to store the resulting sequences in memory.

5. PATTERN EXAMPLE

The following example showcases how patterns can be utilized to create musical complexity from simple parts, using repetition and interference:

```
"<0 2 [4 6](3,4,1) 3*2>".scale('D minor')
.off(1/4, scaleTranspose(2))
.off(1/2, scaleTranspose(6))
.legato(.5)
.echo(4, 1/8, .5)
.tone((await piano()).chain(out()))
.pianoroll()
```

The pattern starts with a rhythm of numbers in mini notation, which are interpreted inside the scale of D minor. Without the scale function, the first line can be expressed as:

```
"<d3 f3 [a3 c3](3, 4, 1) g3*2>"
```

This line could also be expressed without mini notation:

```
slowcat(d3, f3, [a3, c3].euclid(3, 4, 1), g3.fast(2))
```

- slowcat: play elements sequentially, where each lasts one cycle
- brackets: elements inside brackets are divided equally over the time of their parent
- euclid(p, s, o): place p pulses evenly over s steps, with offset o, see https://taogaede.com/wp-content/uploads/2020/01/Research-Paper-on-Euclidean-Rhythms-Aug.-2018-Edit.pdf (cite)
- fast(n): speed up by n. `g3.fast(2)` will play g3 two times.
- off(n, f): copy each event, offset it by n cycles and apply function f
- legato(n): multiply duration of event with n
- echo(t, n, v): copy each event t times, with n cycles in between each copy, decreasing velocity by v
- tone(instrument): play back each event with the given Tone.js instrument
- pianoroll(): visualize events as midi notes in a pianoroll
- Description of structure of demo
- Links to examples/existing tutorial etc

6. TECHNICAL REQUIREMENTS

Space for one laptop + small audio interface (~20 cm x 20cm), with mains power. Stereo sound system, either placed behind presenter (for direct monitoring) or with additional stereo monitors. Audio from audio interface: stereo pair 6,3mm jack outputs (balanced?) good question :) * Projector / screen (HDMI.)

7. ACKNOWLEDGMENTS

Thanks to the Strudel and wider Tidal, live coding, webaudio and free/open source software communities for inspiration and support. Alex McLeans work on this project is supported by a UKRI Future Leaders Fellowship [grant number MR/V025260/1].

8. REFERENCES

Algorithmic Pattern. 2020. Birmingham UK. https://zenodo.org/record/4299661.

Charlie, Roberts, and Kuchera-Morin Joann. 2012. GIBBER: LIVE CODING AUDIO IN THE BROWSER. *International Computer Music Conference Proceedings* 2012. https://quod.lib.umich.edu/i/icmc/bbp2372.2012.011/2/%E2%80%93gibber-live-coding-audio-in-the-browser?page=root;size=150;view=text.

Feedforward. 2020. Birmingham. https://zenodo.org/record/6353969.

Ogborn, David, and J. Beverley. 2017. Estuary: Browser-Based Collaborative Projectional Live Coding of Musical Patterns. *Www.semanticscholar.org*. https://www.semanticscholar.org/paper/Estuary%3A-Browser-based-Collaborative-Projectional-Ogborn-Beverley/c6b5d34575d6230dfd8751ca4af8e5f6e44d916b.

TidalVortex Zero. 2022. Limerick, Ireland. https://zenodo.org/record/6456380.