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 Strudel (or sometimes StrudelCycles), an alternative implementation of the Tidal (or TidalCycles) live coding system, using the JavaScript programming language. Strudel is an attempt to make live coding more accessible, by creating a system that runs entirely in the browser, while opening Tidals approach to algorithmic patterns (Mclean 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 (Open Sound Control) messages, Web Serial and Web MIDI. The project is split into multiple packages, allowing granular reuse in other applications. Apart from TidalCycles, Strudel draws inspiration from many prior existing projects like TidalVortex (McLean et al. 2022), Gibber (Roberts and Kuchera-morin 2012), Estuary (Ogborn et al. 2017), Hydra (Jack [2022] 2022), Ocarina (Solomon [2021] 2022) and Feedforward (McLean 2020).

2. PORTING FROM HASKELL

The original 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 always clear that 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; McLean et al. (2022)], and we have now 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.

3. REPRESENTING PATTERNS

Patterns are the essence of Tidal. Its patterns are abstract entities that represent flows of time as functions, 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 time span 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

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

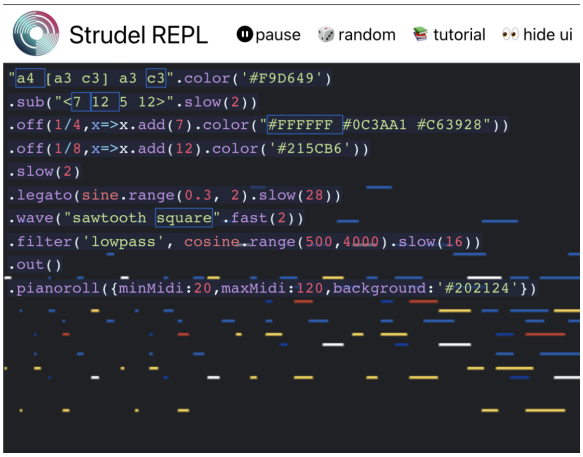

Figure 1: Screenshot of the Strudel editor, including pianoroll visualisation.

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/combination 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))
```

Here is a short description of all the functions used:

- `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 (Toussaint 2005)
- `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

6. WAYS TO MAKE SOUND

To generate sound, Strudel supports different outputs:

- Tone.js
- Web Audio API
- WebDirt, a js recreation of Tidals *Dirt* sample engine
- OSC via osc-js
- MIDI via WebMIDI

Tone.js proved to be limited for the use case of Strudel, where each individual event could potentially have a completely different audio graph. While the Web Audio API takes a *fire-and-forget* approach, creating a lot of Tone.js instruments and effects causes performance issues quickly. For that reason, we chose to search for alternatives.

Strudels Web Audio API output creates a new audio graph for each event. It currently supports basic oscillators, sample playback, envelopes, filters and an experimental support for soundfonts.

WebDirt (Ogborn [2016] 2022) was created as part of the Estuary Live Coding System (Ogborn et al. 2017), and proved to be a solid choice for handling samples in Strudel as well.

Using OSC, it is possible to send messages to SuperDirt (*SuperDirt* [2015] 2022), which is what Tidal does to generate sound. The downside of using OSC is that it requires the user to install SuperCollider and its sc3plugins library, which can be difficult.

The MIDI output can be used to send MIDI messages to either external instruments or to other programs on the same device. Web MIDI is currently only supported on Chromium-based browsers.

7. FUTURE OUTLOOK

The project is still young, with many features on the horizon. As general guiding principles, Strudel aims to be

1. accessible
2. consistent with Tidals approach to pattern
3. modular and extensible

For the future, it is planned to integrate alternative sound engines such as Glicol (Lan [2020] 2022) and Faust (*Faust - Programming Language for Audio Applications and Plugins* [2016] 2022). To improve compatibility with Tidal, more Tidal functions are planned to be ported, as well as full compatibility with SuperDirt. Besides sound, other ways to render events are being explored, such as graphical, and choreographic output. We are also looking into alternative

ways of editing patterns, including multi-user editing for network music, parsing a novel syntax to escape the constraints of javascript, and developing hardware/e-textile interfaces.

8. LINKS

The Strudel REPL is available at https://strudel.tidalcycles.org, including an interactive tutorial. The repository is at https://github.com/tidalcycles/strudel, all the code is open source under the GPL-3.0 License.

9. 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].

References

Faust - Programming Language for Audio Applications and Plugins. (2016) 2022. C++. GRAME. https://github.com/grame-cncm/faust.

Jack, Olivia. (2022) 2022. *Hydra*. https://github.com/ojack/hydra.

Lan, Qichao. (2020) 2022. *Chaosprint/Glicol*. Rust. https://github.com/chaosprint/glicol.

Mclean, Alex. 2020. Algorithmic Pattern. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 265--270. Birmingham, UK. https://zenodo.org/record/4813352.

McLean, Alex. 2020. Feedforward. In *Proceedings of New Interfaces for Musical Expression*. Birmingham. https://zenodo.org/record/6353969.

McLean, Alex, Raphaël Forment, Sylvain Le Beux, and Damián Silvani. 2022. TidalVortex Zero. In *Proceedings of the 7th International Conference on Live Coding*. Limerick, Ireland: Zenodo. https://doi.org/10.5281/zenodo.6456380.

Ogborn, David. (2016) 2022. *Dktr0/WebDirt*. JavaScript. https://github.com/dktr0/WebDirt.

Ogborn, David, Jamie Beverley, Luis Navarro del Angel, Eldad Tsabary, and Alex McLean. 2017. Estuary: Browser-Based Collaborative Projectional Live Coding of Musical Patterns. In *Proceedings of the International Conference on Live Coding*, 11. Morelia.

Roberts, Charles, and Joann Kuchera-morin. 2012. Gibber: Live Coding Audio in the Browser. In *In Proceedings of the 2012 International Computer Music Conference*.

Solomon, Mike. (2021) 2022. *Purescript-Ocarina*. PureScript. https://github.com/mikesol/purescript-ocarina.

SuperDirt. (2015) 2022. SuperCollider. musikinformatik. https://github.com/musikinformatik/SuperDirt.

Toussaint, Godfried. 2005. The Euclidean Algorithm Generates Traditional Musical Rhythms. In *In Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, 4756. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.231.