

Strudel

Felix Roos
Affiliation?
x@x.com

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

ABSTRACT

Abstract goes here (find me in the latex template)

1. INTRODUCTION

This paper introduces Strudel, an alternative implementation of the TidalCycles live coding system, using the JavaScript programming language.

2. BACKGROUND

General motivations / related work. Reference vortex paper and summarise its background.

The reimplementation of TidalCycles in Python (cite TidalVortex) showed that it is possible to translate pure functional reactive programming ideas to a multi paradigm language. It proved to be a stepping stone to move to other multi-paradigm languages, like JavaScript. A significant part of the Python codebase could be ported to JavaScript by syntactical adjustments.

3. INTRODUCING TIDALSTRUDEL

(do we want to call it TidalStrudel once, and Strudel for short from then on as with vortex? Or just stick with Strudel? Should we start calling TidalCycles just Cycles?? froos: I think TidalStrudel sounds a bit weird, but we can stick to the TidalX naming scheme if that's important. For me, StrudelCycles sounds better, because it has 3/4 phonems in common with TidalCycles)

- Motivating musical example

4. TIDAL PATTERNS

(should we explain shortly what tidal patterns do in general here?)

The essence of TidalCycles are Patterns. Patterns are abstract entities that represent flows of time. 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 where the events are placed. From now on, this process of generating events from a time span will be called **querying**. Example:



Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** owner/author(s).

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

© 2022 Copyright held by the owner/author(s).

```
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 defaults to one second, but could be any number of seconds. The console output looks like this:

```
(0  -> 1/2 c3)
(1/2 -> 3/4 e3)
(3/2 -> 1  g3)
```

In this output, each line represents one event. The two fractions represent the begin and end time of the event, followed by its value. In this 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 sequence. In our example `e3` and `g3` are divided equally over the second half of the whole sequence.

5. MINI NOTATION

In this example, the Pattern is created using the `mini` function, which parses Tidals Mini Notation. The Mini Notation is a Domain Specific Language (DSL) that allows expressing rhythms in a short manner.

- Some comparisons of -Strudel with -Vortex and -Cycles code?

(the following examples are from vortex paper, with added js versions)

5.1 1

```
sound "bd ~ [sd cp]"
sound("bd", silence, ["sd", "cp"])
sound("bd ~ [sd cp]")
```

without mini notation:

```
sound $ cat
  [pure "bd", silence,
   cat(pure "sd", pure "cp")]
sound('bd', silence, cat('sd', 'cp'))
```

5.2 2

```
sound "bd ~ <sd cp>"  
sound("bd", silence, slowcat("sd", "cp"))  
sound("bd ~ <sd cp>")  
// sound('bd', silence, slowcat('sd', 'cp'))
```

5.3 3

```
sound "bd {cp sd, lt mt ht}"  
sound("bd", pm(["cp", "sd"], ["lt", "mt", "ht"]))  
?  
?
```

5.4 4

```
sound "bd {cp sd, [lt mt, bd bd] ht}"  
sound("bd", pm(["cp", "sd"],  
[pr(["lt", "mt"],  
["bd", "bd", "bd"]  
)  
,"ht"]))  
??
```

5.5 5

```
sound "bd sd cp" # speed "1 2"  
sound("bd", "sd", "cp") >> speed (1, 2)  
sound("bd sd cp").speed("1 2")  
(operator overloading like in vortex?)
```

5.6 6

```
rev $ sound "bd sd"  
rev(sound("bd", "sd"))  
sound("bd", "sd").rev()  
rev(sound("bd sd"))  
sound("bd sd").rev()
```

5.7 7

```
jux rev $ every 3 (fast 2) $ sound "bd sd"  
jux(rev, every(3, fast(2), sound("bd", "sd")))  
sound("bd", "sd").every(3, fast(2)).jux(rev)  
jux(rev, every(3, fast(2), sound("bd sd")))  
sound("bd sd").every(3, fast(2)).jux(rev)  
(partial application)
```

5.8 8

```
n ("1 2 3" + "4 5") # sound "drum"  
n (sequence(1,2,3) + sequence(4,5)) >> sound "drum"  
n("1 2 3".add("4 5")).sound("drum")  
n("5 [6 7] 8").sound("drum")  
(operator overloading?)
```

5.9 9

```
speed("1 2 3" + sine)  
speed(sequence(1,2,3) + sine)  
speed("1 2 3".add(sine))  
"c3*4".add(sine.mul(12).slow(8)).pianoroll()
```

5.10 10

- Mininotation

6. STRUDEL/WEB SPECIFICS

Some discussion about whether strudel is really a port of TidalCycles, or whether javascript affordances mean its going its own way..

- Recursive Scheduling: calling itself in the future
- Optimizing Syntax for minimal keystrokes / readability: AST Hacking via shift-ast pseudo variables
 - Handling mininotation - double quoted and template strings to mini calls
 - Operator overloading
- Fixing inconsistencies (e.g. with stut/echo) adding source locations
- Dynamic HUD: Highlighting + drawing
- Translation of Tidal concepts to Javascript - different constraints, affordances, aesthetics
- Dynamic Harmonic Programming?
- emulating musician thought patterns
- microtonal features? webserial

6.1 User Code Transpilation

(compare user input vs shifted output)

6.1.1 double quotes -> mini calls

```
"c3 e3" // or `c3 e3`
```

```
mini("c3 e3")
```

6.1.2 operator overloading

```
cat(c3, e3) * 4
```

```
reify(cat("c3", "e3")).fast(4)
```

(reify is redundant here, the shapeshifter could have an additional check)

(TBD: ability to multiply mini notation strings)

6.1.3 pseudo variables

```
cat(c3, r, e3)
```

```
cat("c3", silence, "e3")
```

6.1.4 locations

```
cat(c3, e3)

cat(
  reify("c3").withLocation([1,4,4],[1,6,6]),
  reify("e3").withLocation([1,8,8],[1,10,10])
)

mini("c3 e3")
```

with locations:

```
// "c3 e3"
mini("c3 e3").withMiniLocation([1,0,0],[1,7,7])
```

(talk about mini adding locations of mini notation parser)

6.1.5 top level await

```
const p = (await piano()).toDestination()
cat(c3).tone(p)

(async()=>{
  const p = (await piano()).toDestination();
  return cat("c3").tone(p);
})()
```

7. MUSICAL EXAMPLES

8. ONGOING WORK/FUTURE AIMS

- WASM Sound Backend
- OSC -> SuperCollider
- mininotation as the regex of metre

That Roberts (2016) are excellent, I reference their work at least twice per sentence (Roberts 2016, 3). Another reference (Alternate Timelines for TidalCycles 2021).

"1 2 3"

9. REFERENCES

- gibber
- krill
- glicol

Alternate Timelines for TidalCycles. 2021. Valdivia, Chile.
<https://zenodo.org/record/5788732>.

Roberts, Charles. 2016. Code as Information and Code as Spectacle. *International Journal of Performance Arts and Digital Media* 12 (2): 2016. <https://doi.org/10.1080/14794713.2016.1227602>.