Then Try This • Algorithmic Pattern Salon

Rhythm, Time and Geometry

Xavier Góngora

Then Try This

Published on: Nov 11, 2023

DOI: https://doi.org/10.21428/108765d1.e65cd604

Abstract

The rationale behind RTG, a Haskell library in early development, is presented. RTG aims to provide a tool for the generation and manipulation of rhythmic patterns in music using the concept of geometric structure and transformation as a core abstraction of its API.

Rhythm, Time and Geometry

RTG's development is part of my PhD research into programming languages as musical instruments. My thesis aims to contribute knowledge about the relation between computational abstraction mechanisms and musical expressiveness in the case of the Haskell programming language, focusing on the musical domain of rhythm. This library is intended as a proof of concept and a tool to generate, transform and combine rhythmic patterns leveraging geometric methods.

Specification and Design

The motivation behind RTG comes from my mathematical background as an undergraduate [1]. My thesis advisor Micho Durdevich introduced me to Quantum Geometry:[1] a mathematical theory that exploits the duality between symmetry and geometric structure, heavily influenced by the Erlangen Program (see next section). Inspired by this idea and my practice with the Tidal Cycles live coding system, I came up with three design criteria for RTG's specification which I'll clarify in the next sections:

1. Rhythmic patterns are generated in a declarative way from a collection of algorithms with geometric interpretation.
2. This patterns are organized into families represented by algebraic data types with binary operators defined to make them *groups*.
3. Library modules are divided into three classes: rhythm, time and geometry.

An Interpretation of Geometry

On this section I give a brief account on the concept of geometry that informs RTG's approach. First, lets consider the Erlangen Program's definition of geometry:

> Geometry is the science which studies the properties of figures preserved under the transformations of a certain group of transformations, or, as one also says, the science which studies the invariants of a group of transformations [2].

In these context a group of transformations is also called a symmetry group. The Erlangen Program establishes a duality between geometric structure (such as point configurations, metric and dimension) and algebraic structure (their associated symmetry groups). It is the heritage of Felix Klein, whose work [3] had a decisive

influence on the mathematics and physics of the twentieth century [4]. From this a close relationship between the notion of symmetry and the abstract group definition comes forth and calls for any group to be thought as the set of symmetries of an implicit geometric object. What is formally defined as “the group structure” is, in broad terms, the ability of combining any two transformations to obtain another valid transformation (by means of a binary operator) and being able to revert the effect of any transformation as if nothing had happened.[2]

During the conception of my research project, the capacity of Tidal Cycles to express pattern combination (through the mini-notation and arithmetic operators)[3] and the inherently geometric nature of euclidean rhythms (which can be modeled as point configurations on the circle maximizing an evenness criteria [5]) made me wonder about the inner geometry of rhythmic patterns when seen as transformations (group elements). In which cases do rhythmic patterns have a group structure? If none is found intrinsically, how can we provide it externally? Can this structure have relevant and coherent musical consequences and application? The fact that Tidal Cycles is a library of pure functions, as it is implemented in Haskell [6], gives some plausibility to this intuition, considering the set of functions `f :: a -> b` (from some type `a` to some type `b`) can inherit a group structure in two general circumstances:

1. When its codomain[4] `b` has itself a group structure. In Haskell, this is equivalent to the type `b` having an instance of the type class `Group`.
2. When a set of functions range over its own domain (`b == a` so that `f :: a -> a`) function composition works as a binary operator with the identity function as the operation identity.

The most prominent component of a group is its defining operation. Each pattern arithmetic operator in Tidal Cycles , for instance `*|` or `|+|`, refers to a binary operator of cycle-wise onset events, with patterns `“1”` and `“0”` as identity elements respectively. The elusive property of the group structure is, in general, the existence of inverses. I took the task of identifying or assigning a group structure (by defining an operator that fulfills the group axioms) for each given set of rhythmic patterns. This poses an immediate question: What musical meaning does the inverse of a rhythmic pattern has? Is it to play it backwards in time?

Euclidean Rhythms

My test implementation for an euclidean rhythm group operation, `<+>,` uses modular arithmetic on their symbolic representation. So in this case the inverse (see line 55) is a kind of complement.

```
module Sound.RTG.Geometria.Euclidean (Euclidean, e, (<+>)) where

import Data.Group
import Sound.RTG.Ritmo.Bjorklund (euclideanPattern)

data Euclidean = Euclidean Onsets Pulses Position deriving (Ord)

type Onsets = Int
type Pulses = Int
type Position = Int

instance Eq Euclidean where
```

```
stdForm (k, n, p) == stdForm (k', n', p')

stdForm :: (Int, Int, Int) -> (Int, Int, Int)
stdForm (k, n, p) = (k', n', p')
  where
    k' = k `mod` n'
    n' = abs n
    p' = p `mod` n'

instance Show Euclidean where
  show (Euclidean k n p) = show $ rotateLeft p $ euclideanPattern k n

rotateLeft :: Int -> [a] -> [a]
rotateLeft _ [] = []
rotateLeft n xs = zipWith const (drop n (cycle xs)) xs

-- Combines two euclidean rhythms using modular arithmetic
-- on the least common multiple of pulse granularity.
(<+>) :: Euclidean -> Euclidean -> Euclidean
Euclidean k n p <+> Euclidean k' n' p'
  | (n /= 0) && (n' /= 0) =
    Euclidean ((k + k') `mod` grain) grain ((position + position') `mod` grain)
  | otherwise = error "No defined semantics for zero pulse euclidean rhythms"
  where
    grain = lcm n n'
    position =
      let scaleFactor  = grain `div` n
       in (p `mod` n)   * scaleFactor
    position' =
      let scaleFactor' = grain `div` n'
       in (p' `mod` n') * scaleFactor'

infixl 5 <+>

instance Semigroup Euclidean where
  a <> b = a <+> b

instance Monoid Euclidean where
  mempty = Euclidean 0 1 0

instance Group Euclidean where
  invert (Euclidean k n p) = Euclidean (-k) b (-p)

-- The interface function to construct euclidean rhythms
e :: (Int, Int, Int) -> Euclidean
e (k, n, p) = Euclidean k n p
```

Here euclidean rhythms are expressed by triplets of numbers `e(k,n,p)` where `k` is the number of sound onsets, `n` is the underlying meter and `p` the pattern position (considering rotations by that number of pulse steps).[5] Then the operation works as follows: `e(3,8,1) <+> e(4,13,3)` gives `e(7,104,37)`, `e(7,12,5) <+> e(0,3,0)` gives `e(7,12,5)` and `e(7,12,5) <+> e(3,8,1)` gives `e(10,24,13)`.

The associated patterns are generated by my implementation of the Björklund algorithm [5], which is also used to print them in `stdout` as defined by the show function in line 24 of the previous code block.

```
module Sound.RTG.Ritmo.Bjorklund (euclideanPattern) where

euclideanPattern :: Int -> Int -> [Int]
euclideanPattern onsets pulses = bjorklund front back
  where
    front = replicate onsets [1]
    back = replicate (pulses - onsets) [0]

bjorklund :: [[Int]] -> [[Int]] -> [Int]
bjorklund front back
```

```
| otherwise = concat (front ++ back)
  where
    newFront = zipWith (++) front back
    newBack = diffList front back

-- Auxiliar function for bjorklund
diffList :: [a] -> [a] -> [a]
diffList xs ys
  | lx > ly = drop ly xs
  | otherwise = drop lx ys
  where
    lx = length xs
    ly = length ys
```

Unfortunately, even thought the `<+>` operation relates directly to the symbolic representation of the rhythms, elements of the general form `e(0,n,0)`, for any natural number `n`, all have the identity property for specific elements. This implies both the identity element and inverses are not unique, so the group axioms don't hold. My attempts to solve this made me consider a series of possible strategies to test: have an exhaustive case treatment, using Haskell language extensions to allow modular integer data types, try non-standard euclidean rhythm interpretations to allow for negative values and the use of equivalence relations. In RTG I've taken the challenge of defining `Group` instances for its rhythmic pattern data types.

Rhythmic Patterns Types

A low level computational definition of a rhythmic pattern is that of an isochronous interval set of pulses marked either as onset or rest. This is usually represented as a binary sequence [7]. As a first approach to implementation following the first design criteria, rhythmic patterns are generated with functions that produce such a binary sequence (like the `euclideanPattern` function from the last section).

In reference to my second design criteria, I will define types of rhythmic patterns based on the associated geometric/algorithmic methods. Euclidean rhythms, as shown in the previous section, are the paradigm of such a rhythmic pattern type. Perfectly-balanced and well-formed rhythms as found in Xronomorph [8] are other types I'm currently researching. As a bonus, a collection of predefined patterns accessible by name corresponding to traditional rhythms and musical scales will be included.

The goal then is creating exciting and unusual patterns by means of operators related to the inner structure or representation of the rhythmic pattern types, by writing code such as:

- `play clap $ e(3,8,1) <+> e(5,7,0) <?> claveSon`
- `play guitar $ e(7,12,0) # notes (japanesePentatonic <*> e(7,13,0))` [6]

Note how each rhythmic pattern can be thought as a pattern transformation by partially applying the operators.[7]

Each type of rhythmic pattern has a three part implementation, in reference to the third design criteria. First we have its *geometric* part, where it is defined as a algebraic data type with a `Group` instance, using the syntactic and mathematical facilities of the Haskell type system [6][9][10]. Second, a *rhythmic* part where the algorithm

called to produce the pattern is defined. The code excerpts from the previous section are the geometric and rhythmic parts for euclidean rhythms respectively. Third, the *temporal* part has to do with computational resources and the interaction with an audio server.

A time for everything

As a music creation and experimentation tool, RTG will encode an exploration strategy of the space of rhythmic patterns [11]. At its core, this strategy depends essentially on the operator defined for every `Group` instance. The use of this group operations is the characteristic method in RTG for the discovery and derivation of new rhythmic patterns. My approach to their definitions is based on mathematically and musically informed heuristics. This might be summarized by the following principles:

1. The definition should have an intrinsic relation with the rhythmic pattern structure or representation.
2. The results must be musically significant.
3. Be as simple as possible (Occam's razor).

An intrinsic relation between the pattern structure and operation can be showed by mathematical and syntactic arguments. On the other hand, simplicity is approached in broad and informal terms. The problematic principle is the second: what criteria will be used to account for musical significance? In the case of rhythm, G. Toussaint describes measures and properties that correlate with the perception of a "good rhythm", which is broadly characterized by adoption and ubiquity in cultural traditions [12]. This approach approach could be used to refine my group definitions and (in consequence) the exploration strategy. This poses an exciting landscape for experimentation, to be informed also by artistic and aesthetic considerations.

Interdisciplinary research into musical rhythm has focused traditionally on the purely temporal aspect, and this project expects to leverage the insights gained around it. I recognize that other dimensions of music, and the body as an active participant in its cognition and generation, play a crucial role in the perception of rhythm [13]. RTG approaches the temporal dimension of music using geometrical ideas as a means to generate and manipulate rhythm towards extending expressiveness at the pattern level. This is because I believe

> “the description of music in terms of a transformationally oriented conceptual vocabulary will help evolve a more appropriate vocabulary and syntax for the description, understanding, and creation of experiences in time, for all self-referential temporal arts, of which music is a very pure example” [14].

Work towards extending this approach to other musical dimensions (such as pitch, dynamics and musical form) is on the roadmap. In the case of pitch one approach is to use the isomorphism between equally tempered scales and rhythmic patterns, understood as a binary sequences of isochronous onsets and rests as mentioned before.[8]

Final thoughts

The search for geometric structure in rhythmic patterns is informed by the Erlangen Program and the possibilities of pattern combination in Tidal Cycles. This is encoded into the definition of `Group` instances. The inner structure of the rhythmic pattern types is of a more conventional geometric nature. Examples of this types, to be implemented in RTG, are euclidean, perfectly balanced and well-formed rhythms, all derived from particularly rich and simple principles that produce engaging patterns. Gathering all this, my intent is to further explore new strategies for pattern transformation.

A visual component might be missed from this discussion. This is intentionally so because so far I'm interested specifically with the interplay of structure and rhythm in a programming language context. In a live coding performance, for example, an euclidean rhythm can be declared in code without being displayed visually without any apparent lose: what is gained is musical expressiveness of the pattern language. Of course, a visual representation of the pattern might be both instructive and fun.

This are the first steps towards making this library a platform for developing computational tools that leverage existing research on the geometric nature of rhythm and the algorithmic generation of rhythmic patterns. Much development work is still ahead towards a usable musical tool.

Footnotes

1. Also known in the literature as non-commutative geometry. ↩
2. In mathematics, a group is a non-empty set with an associative binary operation (in programming terms: a function of two values of the same type that produces a value of that same type), a unique (left and right) identity element and inverses for every element. See https://en.wikipedia.org/wiki/Group_(mathematics) ↩
3. See https://tidalcycles.org/docs/reference/pattern_structure ↩
4. In mathematics, the codomain or set of destination of a function is the set into which all of the output of the function is constrained to fall. See https://en.wikipedia.org/wiki/Codomain ↩
5. They are used in Tidal Cycles to define onset patterns like this: d1 $ sound "cp(3,8,0)". The corresponding euclidean rhythm can be represented by the list [1,0,0,1,0,0,1,0], where 1's are onsets or triggers (in this case, of a clap sample), 0's are rests, and the list elements represents pulses of the same duration within a cycle. ↩
6. The hashtag symbol references Tidal Cycles syntax to pass control patterns. Note the combination of a melodic scale name with an euclidean rhythm. I'll comment on this in the next section. ↩
7. This is called *currying*. See https://en.wikipedia.org/wiki/Currying ↩

8. The euclidean rhythm e(7,12,0) is an example of this. It corresponds directly to the interval pattern of the diatonic scale in a 12 tone equal temperament and, according to Toussaint (2005), is also a common West African bell pattern. ↩

References

1. Zúñiga Góngora, Francisco Javier. 2015. “Teoría de Galois y Haces Principales Cuánticos.” Tesis, CDMX, México: Universidad Nacional Autónoma de México. https://hdl.handle.net/20.500.14330/TES01000743847. ↩

2. Yaglom, I. M. 1988. *Felix Klein and Sophus Lie: Evolution of the Idea of Symmetry in the Nineteenth Century*. Edited by Hardy Grant and Abe Shenitzer. Translated by Sergei Sossinsky. Boston: Birkhäuser. p. 115 ↩

3. Klein, Felix. 1983. “A Comparative Review of Recent Researches in Geometry.” *Bulletin of the New York Mathematical Society* 2 (10): 215–49. https://projecteuclid.org/journals/bulletin-of-the-new-york-mathematical-society/volume-2/issue-10/A-comparative-review-of-recent-researches-in-geometry/bams/1183407629.full. ↩

4. Ji, Lizhen, and Athanase Papadopoulos, eds. 2015. *Sophus Lie and Felix Klein: The Erlangen Program and Its Impact in Mathematics and Physics*. IRMA Lectures in Mathematics and Theoretical Physics 23. Zürich: European Mathematical Society. ↩

5. Toussaint, Godfried. 2005. “The Euclidean Algorithm Generates Traditional Musical Rhythms.” *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, 47–56. ↩

6. Hudak, Paul, John Hughes, Simon Peyton Jones, and Philip Wadler. 2007. “A History of Haskell: Being Lazy with Class.” In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, 12-1-12–55. HOPL III. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1238844.1238856. ↩

7. Toussaint, Godfried T. 2020. *The Geometry of Musical Rhythm: What Makes a “Good” Rhythm Good?* Second edition. Boca Raton London New York: CRC Press. p. 3 ↩

8. Milne, Andrew J, Steffen A Herff, David Bulger, William A Sethares, and Roger T Dean. 2016. “XronoMorph: Algorithmic Generation of Perfectly Balanced and Well-Formed Rhythms.” *Proceedings of the International Conference on New Interfaces for Musical Expression*, July, 6. https://doi.org/http://doi.org/10.5281/zenodo.1176082. ↩

9. Hudak, Paul, and Donya Quick. 2018. *The Haskell School of Music: From Signals to Symphonies*. 1st ed. Cambridge University Press. https://doi.org/10.1017/9781108241861. ↩

10. Wadler, P., and S. Blott. 1989. “How to Make Ad-Hoc Polymorphism Less Ad Hoc.” In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 60–76. POPL ’89. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/75277.75283. ↩

11. Wiggins, Geraint A. 2017. “Un marco teórico para la descripción, el análisis y la comparación de sistemas creativos.” In *Creatividad computacional*, edited by Rafael Pérez y Pérez, 2ª reimp. México, D.F: Grupo Editorial Patria Universidad Autónoma Metropolitana-Unidad Cuajimalpa. ↩

12. Toussaint, Godfried T. 2020. *The Geometry of Musical Rhythm: What Makes a “Good” Rhythm Good?* Second edition. Boca Raton London New York: CRC Press. ↩

13. Clarke, Eric F. 1999. “Rhythm and Timing in Music.” In *The Psychology of Music (Second Edition)*, edited by Diana Deutsch, 473–500. Cognition and Perception. San Diego: Academic Press. https://doi.org/10.1016/B978-012213564-4/50014-7. ↩

14. Spiegel, Laurie. 1981. “Manipulations of Musical Patterns.” In *Proceedings of the Symposium on Small Computers and the Arts*, 19–22. http://lauriespiegel.net/ls/writings/musical_manip.html. ↩