Making Music from L-Systems

Steven Schwarz

New Music DC 2022
October 15-16, 2022

Using L-Systems to synchronize line-drawing fractal animations with music

An *L-System* consists of:

- an *alphabet* of *symbols* used to form *strings*
- a set of *replacement rules* of the form **symbol ⇒ string**
- an initial *axiom* symbol

A **symbol** in the alphabet that appears on the left in some *replacement rule* is a *variable*; other symbols are *constants*. Each *variable* will be on the left in one and only one *replacement rule*. A **string** on the right in a replacement rule may contain a mix of *variables* and *constants*.

Each L-System generates a sequence of strings that captures a particular *pattern of fractal self-similarity*, as follows:

- We start with the *axiom* symbol; this is the *generation 0* string.
- If we have the *generation n* string, then the *generation n + 1* string is formed from it. Each *variable* in it is replaced by the **string** on the right in that variable's *replacement rule*. (*Constants* are left alone.)

An L-System may be given a set of *encoding rules* of the form **symbol ⇒ string**, where the **symbol** appearing on the left must be a *variable*, and all symbols in the **string** on the right must be *constants*. Each *variable* will be on the left in one and only one *encoding rule*. A *generation n* string is then encoded, by replacing each *variable* by the **string** on the right in that variable's *encoding rule*. An encoded *generation n* string will consist entirely of *constants*.

An L-System is said to be *designed for visualization* if it has *encoding rules*, and its *constants* are interpretable as commands to a simple drawing engine, such as the *logo turtle*. Imagine a turtle that travels about on the computer screen, taking steps, and making turns, and that can remember a small number of details called its *current drawing state*:

- current *location*
- current *direction*
- current *step size*
- current *turning angle*

A typical set of *logo turtle* commands would include:

l	take one *visible* step
x	take one *invisible* step
+	turn to the *right*
–	turn to the *left*

Other, more complicated, commands might be used, *e.g.* to change the current *step size*. The encoding of a *generation n* string will cause the turtle to trace out a *fractal line drawing* called the *level n visualization* of the L-System.

By making a few *special assumptions*, a correspondence with *visualization* interprets an encoded *generation N* string as a *melody*. The L-Systems I use always give the logo turtle a permanent rotation angle of 30°, and the turtle always starts out headed in the 0° direction. This means that the turtle's current direction can always be converted to a standard twelve-tone pitch class chosen from

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

by dividing the current compass heading by 30 and using arithmetic modulo 12. Similarly, my L-Systems always give the logo turtle *step sizes* that are integers. This means *step sizes* always correspond musical *note* or *rest* durations that can be expressed (possibly with ties) using common practice notations such as *quarter*, *half*, and *whole notes* or *rests*. When the turtle is asked to take a step leaving a *visible* trail, that corresponds to playing a *note*. When the trail is *invisible*, that corresponds to a *rest*.

Here is a simple example L-System:

axiom:
`F`
replacement rules:
`C ⇒ (F----C+F+++C)`
`F ⇒ (F---C-F++++C)`
encoding rules:
`C ⇒ f`
`F ⇒ f`

The alphabet for this simple system contains just two *variables*, `F` and `C`. Some of the turtle commands above appear as *constants*. (Left and right *parentheses* are also used as *constants*. These have no meaning to the turtle, who simply ignores them. They are only there to facilitate reading complex *generation n* strings.)

Below are the first few *generation n* strings, with their encodings:

generation 0	`F`
encoded	`f`
generation 1	`(F---C-F++++C)`
encoded	`(f---f-f++++f)`

generation 2	((F---C-F++++C) --- (F----C+F+++C) - (F---C-F++++C) ++++ (F----C+F+++C))
encoded	((f---f-f++++f) --- (f----f+f+++f) - (f---f-f++++f) ++++ (f----f+f+++f))

Note that each *generation n + 1 string* is at least four times more complicated than the *generation n string*, because each new *generation* applies a *replacement rule* four times.

Here is a hint at the significance of the symbols `F` and `C`. Note that both represent taking a sequence of four steps, each step labeled `F` or `C`, with some number of + and − turns in between. Read the sequence of steps and turns for `F` backwards, substituting `F` for `C` and *vice versa*, and substituting + for − and *vice versa*. The result is the exact sequence of steps and turns for `C` read forwards. `F` then corresponds to the *abstract* notion of a forward step on this L-System's *fractal*, while `C` corresponds to an *abstract* backward step. Each *level n visualization* will show a concrete approximation of the *fractal* formed from these *abstract* steps. There will be progressively more detail for each larger *n*.

This example is an instance of L-Systems that derive from a *fractal line drawing generator*. In full generality, such L-Systems make use of variables `B`, `C`, `F`, and `G`, whose abstract meanings, and turtle interpretations, are shown in the following table:

step type	*meaning*	*turtle interpretation*
`B`	a *backward* step	turtle goes *backward above* the screen
`C`	a *backward* step, turns *reversed*	turtle goes *backward below* the screen
`F`	a *forward* step	turtle goes *forward above* the screen
`G`	a *forward* step, turns *reversed*	Turtle goes *forward below* the screen

Note how this corresponds to *imitations* in 18th-century music, where a *subject* may appear *forwards* or *backwards*, and either type of *imitation* may be *inverted*.

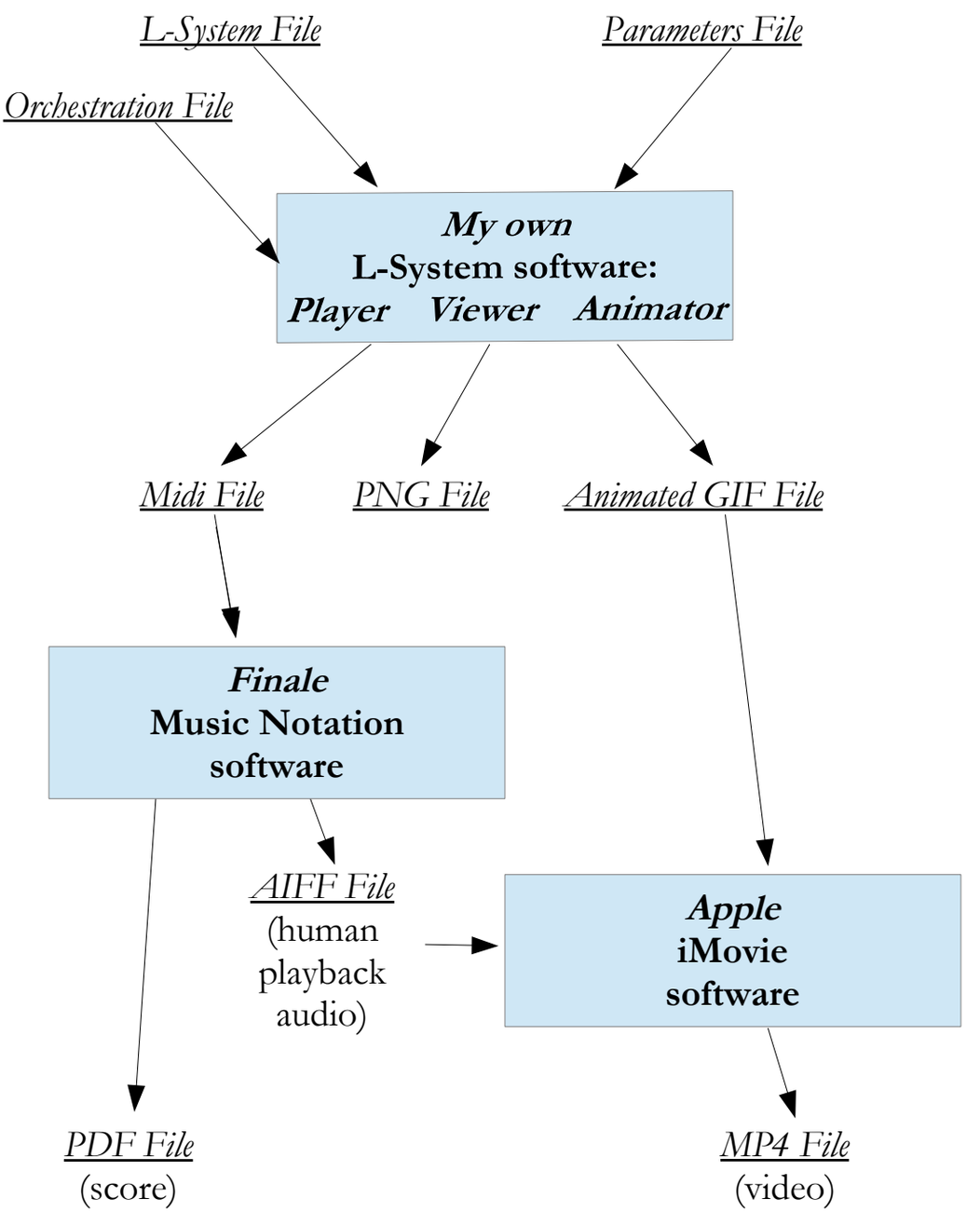
When I use an L-System to generate music, I form *encoded generation strings* from *generation 0* through some *generation n - 1*, and then combine them into an *n-voice nested canon by augmentation*. The *generation 0* string consists of a single note that lasts throughout the entire piece. The notes of each later *generation* become progressively shorter. All the notes that come from using a single *replacement rule* on the previous *generation* string, taken together, last the length of the *note* or *rest* that was replaced.

I typically show *animations* of turtle drawings being made, rather than just a *still image* of the completed drawing. This way, one can enjoy following the exact path the turtle takes. For example, one can take proper note of occasions when the turtle redraws a previously drawn step (possibly with a different color). Only the *final generation* fractal line drawing is *animated*, whereas this animation is synchronized to a soundtrack that is made from the *canon* of all the *generations*. The reasons for this choice are complicated and have to do with how our sense of vision and hearing react differently to *superposed stimuli*.

Here's how I use *color* in these animations:

- Colors for steps that correspond to *notes* are chosen randomly – avoiding grayscale colors.
- All steps corresponding to *rests* get a single color – a medium gray.
- Color choices have no inherent aesthetic significance (to me at least).
- All steps taken in the *encoded generation n* string that trace back to one step in the *encoded generation 1* string will have the same color (but recall that steps that correspond to *rests* are always colored *gray*).
- Each step taken is colored *white* as it is first made (whether it is for a *note* or a *rest*). When the next step is taken, the previous step's *white* trace will change to the color it is meant to have.
- The starting (*resp.* ending) location of the Turtle is marked by a *green* (*resp. red*) dot. In case these locations are the same (or nearly the same), the *green* dot will be drawn larger than the *red* dot, so that both may be seen.
- Everything takes place against a *black* background.

My composition workflow is outlined on the following page. Colored boxes show pieces of software, some standard, and some written by me. Inputs and outputs are connected to the relevant software by arrows.



Composition Workflow

There are many other types of L-Systems than those derived from fractal line drawing *generators*.

In fact, L-Systems were initially developed to model the *branching structures* of plants. *Branching* L-Systems typically use constants [and]. When the logo turtle encounters the [command, it *pushes* a copy of its current drawing state onto a *stack* of saved drawing states, and continues to process drawing commands, which causes the current drawing state to evolve as usual. At some later point, when the turtle encounters the] command, it *pops* the top saved drawing state off the top of the *stack*, adopting its details into its current drawing state. This allows the turtle to follow one branch, and then return to the branching point to follow another. Here is a typical example of the type of plantlike branching structure one can capture with a *branching* L-System.

Another flavor of L-System bases each L-system on a particular *Sturmian word*. A *Sturmian word* is a never-ending sequence of 0s and 1s, with a tantalizing property: the sequence never settles down to just repeating some finite sub-sequence, but the sequence has just enough complexity built into it – and no more – to accomplish that. A *Sturmian word* is thus the simplest possible *transcendental* object.[1] It turns out that each *Sturmian word* can be used to form an L-System whose variables are `0` and `1`. The *Sturmian word* itself is the *generation* ∞ *string* of the L-System. Here is a typical *Sturmian word* L-System:

axiom:

```
1
```

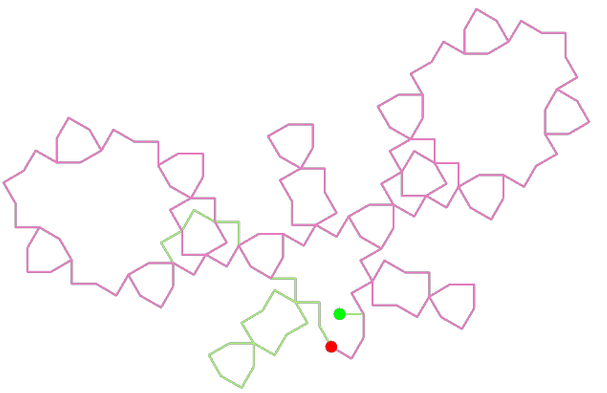
replacement rules:

```
0 ⇒ (10010)
1 ⇒ (10)
```

encoding rules:

```
0 ⇒ f+
1 ⇒ f|||
```

Note that the *replacement rules* make use of *no constants* (other than parentheses, which the turtle ignores). Also note that the rule for `0` takes 5 steps, but the rule for `1` takes only 2. This will translate to music that frequently switches between 2/4 and 5/4 *time signatures*, switches between 2-measure and 5-measure phrases, switches between 2-phrase and 5-phrase phrase groups, and so on. Finally, a new graphics command to the logo turtle makes an appearance in this L-System: `|`. To handle this `|` command, the logo turtle must add a *step serial number* to its drawing state, and each time it makes a *step*, the *step serial number* is incremented by one. When the `|` command is encountered, the turtle looks at the current *step serial number*, and turns *right* or *left* according as the number is *even* or *odd*. So, for each `0`, the turtle takes a step and turns right, but for each `1`, the turtle takes a step and turns right or left according to the parity of the *step serial number* after taking that step. Here is what a still image of the turtle's *generation 5* line drawing looks like:



[1] The simplest characterization of a *Sturmian word* make use of the collection of all finite sub-strings of 0s and 1s that may be found somewhere in the infinite word. Such a collection is called a *language*. In the *language* that corresponds to a *Sturmian word*, the number of different words of length *n* is always precisely *n + 1*. Clearly, if there were some length *n* for which there were fewer than *n + 1* different words of that length in the language, then the word could not be *Sturmian*, because it would have to repeat sequences of length *n* forever from some point on. It turns out the converse is also true, so Sturmian words just correspond to languages that always have *n + 1* different words (and no more) of length *n*.

Yet another type of L-System comes from *single-vertex flat-fold origami*. Imagine doing origami with a piece of circular paper, subject to two restrictions:

- all folds must make a *crease* that extends from the *center* of the paper to the *edge*
- after all folding has taken place, the paper lies *flat*

It turns out that such an origami is completely described by the sequence of angles formed by adjacent creases, circling around the center, and that there must be a sequence of mountain and valley folds one can make, beginning with two mountain folds, and alternating valley with mountain after that, which ends up folding the paper flat. (Of course, one could equally well begin with two valley folds, and then alternate mountain and valley after. This corresponds to looking at the paper from below rather than from above.) This leads to a kind of L-System based on *variables* `M` and `V`, in which the + and − turns made exactly match the angles between adjacent creases. Here is an example:

axiom:

```
M
```

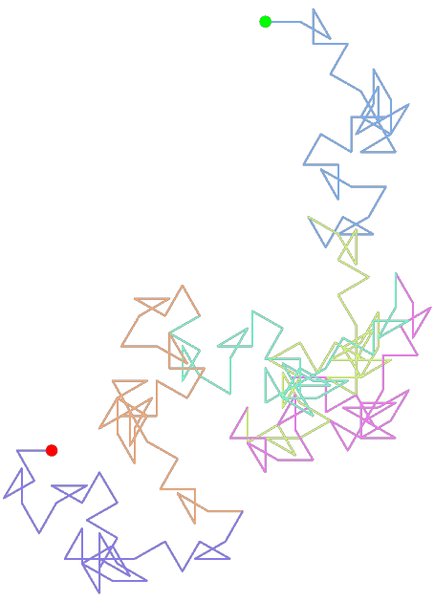
replacement rules:

```
M ⇒ ++++M-M+++++V-------M+++V----M
V ⇒ V++++M---V+++++++M-----V+V----
```

encoding rules:

```
M ⇒ f
V ⇒ f
```

Here is what a still image of the turtle's *generation 2* line drawing looks like:



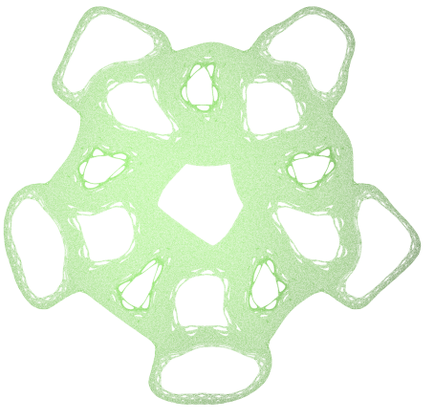
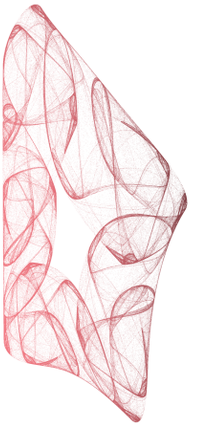
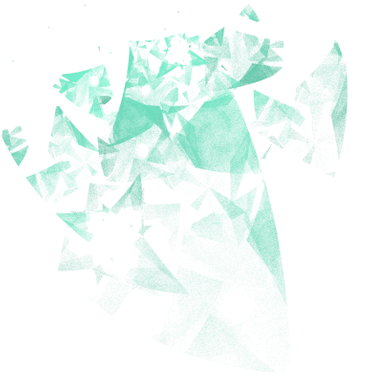
Finally, there are myriad other ways to match graphics animations with music derived from L-Systems. One idea I have developed recently involves using very simple *one-dimensional* L-Systems to sample points from *strange attractors*.

A *strange attractor* is formed when a *chaotic* plane mapping is iterated. In general, when a plane mapping is iterated, there are several possible outcomes:

- there is a *fixed point* that repeats forever

- there is a *finite cycle* of points that repeats forever
- the map is *unstable*, with points diverging off to infinity
- there is *chaos*, where small differences in initial conditions produce large differences in the map

When a plane map exhibits *chaos*, iterating it develops a figure called a *strange attractor*, which is in fact comprised of a sequence of millions of discrete points scattered through the attractor, but often appears to place "other structure" on them. Here are a few examples of strange attractors:

Starting with a strange attractor, and an L-System using graphics constants meant for sampling, one can sample a few hundred to a few thousand of the millions of points of the attractor, producing both an animation and an equivalent musical shape. The basic idea is simple: adjacent points in the sampling determine a line segment connecting them. The endpoints of this segment are on the attractor, but the points in between are typically not. The slope of that line segment may be *quantized* to a twelve-tone pitch (by rounding to the nearest multiple of 30°, dividing by 30, and using arithmetic modulo 12). The duration of the pitch will come from the L-System.

To make an L-System just for sampling, one restricts the logo turtle to using graphics commands that make visible or invisible steps. Here is a possible set of commands (two of which have been used above, and two of them are new):

`f`	*visible* step
`x`	*invisible* step
`m`	step *visible in all but the final generation*
`p`	*placeholder* step (*invisible*, and 0 used as the step size rather than current step size)

One may also provide the turtle with commands that alter the current *step size*. But notice there is no way for the turtle to turn right or left. So these turtles only go *forward in one direction*. They are especially good for sampling through a sequence of iterated plane map points.

To match each musical note or rest derived from a line segment connecting samples, as just explained, the animation will show, against a complete image of the attractor, each segment drawn in white with a green dot marking the previous sample and a red dot marking the current sample. At the end of the animation, all these segments are shown, again in white, with green and red dots marking the first and last samples of the entire sampling sequence. The effect is as though a *spider* skilled in sampling had spun a *web*, thread by thread, using points of the attractor to anchor each thread, and then at the end, the entire *web* is shown shrouding the attractor.

To look at actual music videos made using the ideas explained here, please go to this playlist on my YouTube channel:

https://www.youtube.com/playlist?list=PLrVJvh-lyQaTfgMQ9qU_C-0MQU1OXco8h