Lissajus Curves: an Experiment in Creative Coding

Lali Barrière
Dept. of Applied Mathematics 4, Universitat Politècnica de Catalunya
c. Esteve Terradas 5, C3-D007, 08860 Castelldefels, Spain
lali@ma4.upc.edu

Abstract

Do you want to become a creative coder for a day? When artists use a computer to make art, they often experience the need of improving their mathematical skills. On the other hand, students with technical skills are less comfortable with aesthetic issues and creative work. In this workshop we use Processing to build two animations that involve: (1) the use of trigonometry and polar coordinates in parametric curves, and (2) the (soft) implementation of Hooke's law. Our goal is to show that creative coding is a rewarding way of learning mathematics, with a creative approach.

Introduction

The use of computers for creative purposes began with the onset of general purpose computers, towards the decade of 1950-60. Artists took ownership of this technology from the very beginning. The development of electronic and digital technologies led, over the last century, to the development of existing artistic disciplines, thanks to the new available tools. Musicians, for example, found new tools for the synthesis of new sounds, and for composition and production; as a consequence, musical creation changed from then on. Moreover, new forms of creativity rose up, giving birth of new disciplines such as net art, sofware art, and making conceptual and generative arts to evolve towards computational forms of expression. Creative coding, the use of computer programming to make art, is one of these forms of expression [4].

Besides the facts that mathematics are at the core of computer technology and that computers made tractable mathematical problems that were intractable before, computer artists were able to see the creative potential of these new machines. Indeed, the pioneers of computer art were mathematicians, engineers, or worked closely with them. Nowadays, creative coders with no or little mathematical education often experience the need of learning mathematics, or improving their mathematical skills, overcoming their initial fear of Calculus, Geometry, and Algebra for their artistic purposes. On the other hand, students with technical skills, such as engineering students, are not very comfortable with aesthetic issues and creative work.

Since creative coding involves the creative use of mathematics, we think that it makes their learning attractive and rewarding. But not only this, it also faces those who were comfortable with them to an uncomfortable situation: that of making decisions according to artistic and aesthetic criteria. This is why we think that teaching art and mathematics through creative coding may contribute to make students interested in mathematics, to lead them to collaborative work, and to overcome the usual distinction between students with scientific orientation and those more focused on the arts and humanities, helping to break the old routines that are still present in our educational systems [6].

The aim of this workshop is to demonstrate by example a possible way to do it. We will use the Processing programming language, which has also been used in the workshop [7] by Ruttkay in Bridges 2010.

The Tool: Processing

Processing [5] is a free and open source programming language and development environment for creative coding in a visual context. It was initially created to help artists and designers to learn computer programming with a gentle learning curve. The project began in 2001 at the MIT Media Lab, where Casey Reas and Ben Fry, who are now the directors of the project, studied under the advising of John Maeda. As an open project, a big international community of developers and supporters now maintains Processing, which has grown from a tool for learning and prototyping to a development tool for professional production. The Processing web page [5] is a great source of information as well as many books on programming with Processing, such as the handbook [1] by Reas and Fry, and the somewhat deeper introduction [2] by Greenberg, with a computer scientist approach.

The language adds to Java the primitives and functionalities needed for drawing, animation and interaction. And this, without the complexity of the usual programming environments that drives newbies away. I have been using Processing since 2008 for my creative work and for teaching programming to artists, designers, at Engineering Schools and Fine Arts Schools. After teaching creative coding to a variety of audiences, I am convinced that Processing is the best tool, not only for teaching programming to beginners, but also to put together scientific and creative learning.

The Workshop: Goals and Structure

In this workshop we will generate animations that make use of some mathematics (see Figure 1) and physics (see Figure 2) using Processing.

Regarding the mathematics and physics we will make use of, the *leitmotiv* will be "keep it simple", as we think that we should do less, but do it better. For those who want to delve into the mathematics and physics needed by a creative coder, the books by Shiffman [8] and Kodicek [3] are good overviews.

The workshop is open to anyone interested in creative programming and its relations with mathematics. Despite the obvious time limitations, we intend to follow a path that will lead us from a basic, short introduction to the Processing environment and language to the artworks, step by step. Some programming background may help follow the workshop with ease. On the other hand, those that find mathematics too basic, might be interested in the creative aspects of our work.

Workshop objectives. Our goal is to show a possible way of teaching programming and mathematics for creative purposes in a visual context. Since we work with examples, we will focus on the mathematics and physics needed for these examples. Still, our aim is to stress the interest in the way mathematics, programming and art interact, more than in those specific examples.

More precisely, what attendees can expect from this workshop is:

- To acquire a basic knowledge of the structure of a *sketch*, a program in Processing, and to figure out the wide range of possibilities offered by this language.
- To understand the mathematics and physics that we use in the supplied sketches.
- To be able to follow the sketches, focusing on the part of the code in which mathematics is used.
- To modify the codes or create new ones, with similar (or not so similar) ideas, paying attention to the aesthetic aspects of the artwork.
- To get motivated to go on with creative programming.

In summary, we aim to make a straightforward use of Processing, focusing on the mathematics that define the geometry and the animation of our *sketches*; to work on the aesthetic and artistic aspects of our work; and to understand creative coding as a tool for teaching and learning mathematics and art.

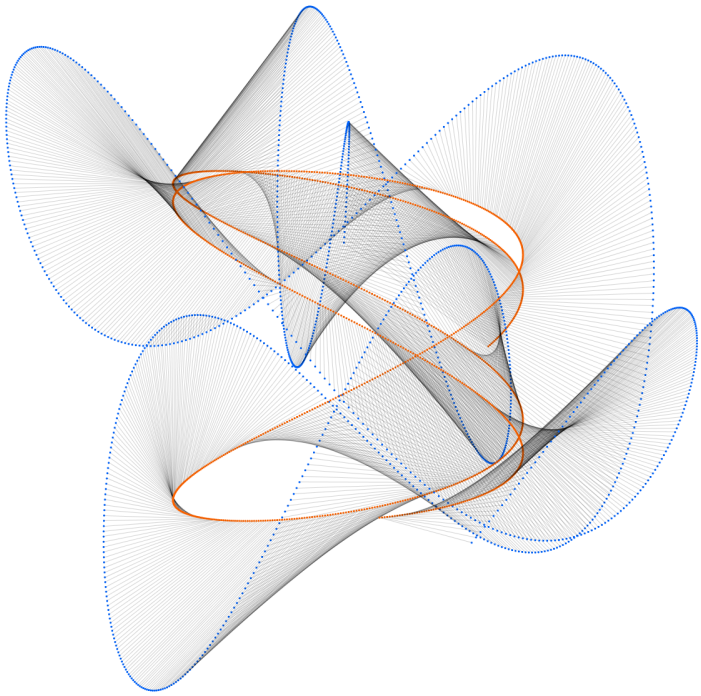

Figure 1 : *Lines connecting a Lissajus curve and a double-Lissajus curve with some random component.*

Workshop organization. All the needed codes will be available to the attendees before the conference at `http://www.lalibarriere.net/bridges/bridges.zip`. The workshop is organized in three parts:

1. We will start with a quick introduction to the language, in which we will first show how the structure of a sketch is and then explain which are the primitives that we need in our artworks. By using some simple codes, we will illustrate every functionality while focusing our attention on the code blocks that make the sketch structured.
2. After introducing Processing we will work on the first artwork. This will be the core of the workshop and will serve as a basis for the second artwork. We will explain in detail how trigonometry is used to parametrize curves, which are then used to implement the animation (see Figure 1).
3. The workshop will finish with the study of the second artwork, which uses the same curve as the first example. The main feature of this sketch is that it makes use of the simple physics of elastic springs to define the graphics and their behavior (see Figure 2).

Workshop needs. There are some practical questions that we think it is worth mentioning.

- Though we say that we will explain the needed mathematics and physics, we assume the knowledge of elementary trigonometry and Newton's second law.
- It might be a good idea to work in pairs, especially for attendees having no programming background.
- As it may seem obvious, a computer is needed to do the practice in this workshop. Processing is available for Mac OS, Linux and Windows platforms and freely downloadable from `http://processing.org/download/`.
- For those interested in using Processing further, we can give pointers to some interesting and helping resources.

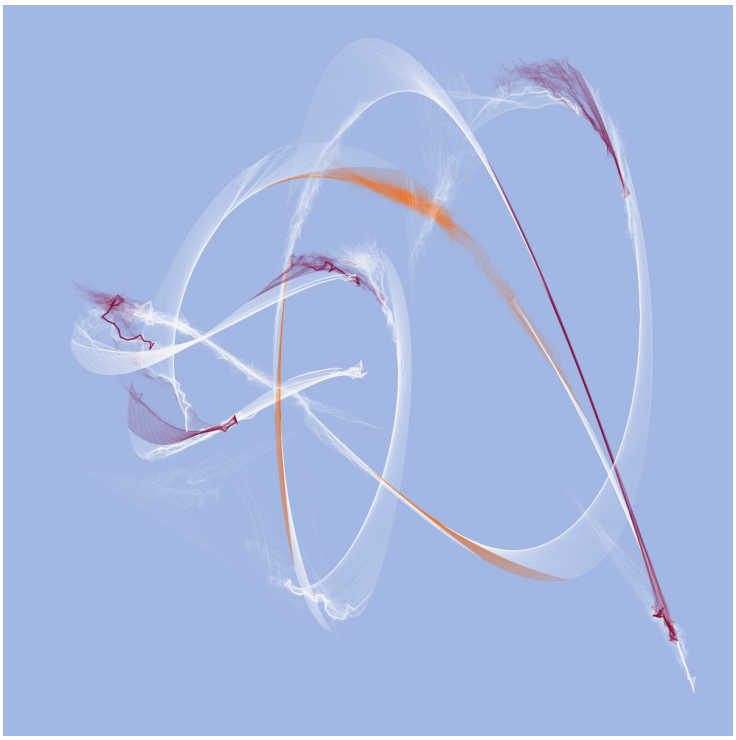

Figure 2 : *A string of particles that looks like a paintbrush.*

The Artworks: Can Artists Deal with Parametric Curves and Hooke's Law?

There are several reasons why we choose these particular examples. First, the aesthetic result is appealing and they can easily be changed to obtain different results. Also, they can be built up from scratch, in a way that allows us to show the structure of the code. Finally, the mathematics involved are basic, but they make part of the background of every digital artist.

Now, let us go into a brief description of the two promised examples.

Parametric curves. Or working with trigonometry and polar coordinates to obtain a beautiful output.

The animation is made of two paths connected by lines. The first path describes a Lissajus curve, while the second one describes what I called a *double-Lissajus curve* (see Figure 3). This choice allows us to avoid some control operations that may disturb the mathematical discourse and, more importantly, makes easy to explain how trigonometry can (indeed, has to) be used to become a good digital artist.

The parametric equation of the standard Lissajus curve is:

$$\left.\begin{aligned} x(t) &= r_1 \cdot \cos(\alpha \cdot t) \\ y(t) &= r_1 \cdot \sin(\beta \cdot t) \end{aligned}\right\}$$

The parametric equation of the double-Lissajus curve is:

$$\left.\begin{aligned} x(t) &= r_1 \cdot \cos(\alpha \cdot t) + r_2 \cdot \cos(a \cdot t) \\ y(t) &= r_1 \cdot \sin(\beta \cdot t) + r_2 \cdot \sin(b \cdot t) \end{aligned}\right\}$$

It can be easily seen that the second curve evolves as a "satellite" of the first one. Also, the use of polar coordinates is quite simple, although it may require some explanation to non-mathematically educated attendees. The implementation of these curves in the sketch is as follows:

```
// the first curve is a Lissajus curve
x1 = radi1*cos(angle*alpha);
y1 = radi1*sin(angle*beta);
// the second curve is a double-Lissajus curve
x2 = x1 + radi2*cos(angle*a);
y2 = y1 + radi2*sin(angle*b);
// drawing points and lines that follow the curves
ellipse(x1, y1, 4, 4);
ellipse(x2, y2, 4, 4);
line(x1, y1, x2, y2);
 // making a step
angle+=step;
```

The right choice of the radii r_1 and r_2 makes the animation stay inside the frame with no need of control, as announced above. In the supplied *sketch*, the parameters α, β, a, and b are randomly chosen. That makes each execution of the sketch create a unique output. This is a standard feature of generative artworks. Notice also that, for the sake of simplicity, the oscillations start in phase.

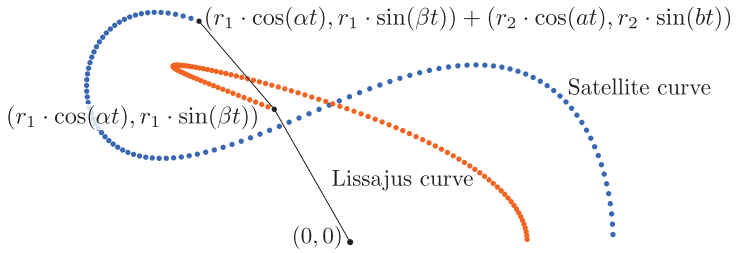

Figure 3 : *Parametric curves for our first* sketch.

Adding physics to our sketch. Or particles following each other under the action of Hooke's law.

After working with curves to build an animation, we make a step further, adding physics to our sketch. In doing this, we stay on an easy, gentle pace, as we only add springs to our curve, simulating the elasticity described by Hooke's law, and dismissing the action of any other force. The usual simple formulation of this law is:

$$F = -k \cdot (x - \ell)$$

where F is the force, k is the spring constant, x is the current length of the spring, and ℓ is the length of the spring at rest.

We define a string of particles connected by a series of springs. The first particle moves following a double-Lissajus curve, the others are followers subject to the force of the springs: the first spring connects the first and second particles, the second spring connects the second and third particles, and so on (see Figure 4). For aesthetical purposes, we add some mouse interaction, which causes distortions on the mathematically defined path.

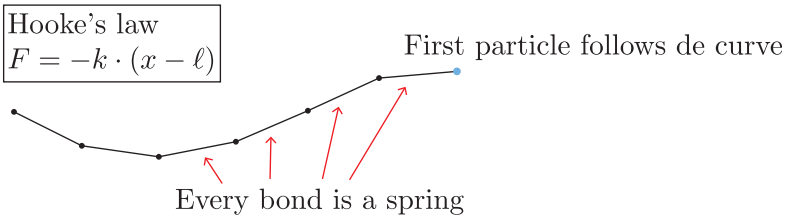

Figure 4 : *The string of particles joined by strings in our second* sketch.

Implementing simulation, either realistic or not, usually involves some technicalities, often hidden in a library of functions. In our case, these functions are simple enough to be included in the codes: a string of particles, which is made up by an array of points together with an array of springs, and the elasticity of the springs, in a function.

The elastic force is implemented in Processing by defining a spring as a pair of particles mutually interacting according to the following function:

```
void calculateForcesOnParticles() {
  PVector diff = new PVector(0, 0);
  diff.x = B.x - A.x;
  diff.y = B.y - A.y;
  float l = diff.mag();
  diff.normalize();
  float force = k * (dist - l);
  A.x -= diff.x * force;
  A.y -= diff.y * force;
  B.x += diff.x * force;
  B.y += diff.y * force;
}
```

The string of particles together with the springs that connect them can be implemented as follows:

```
particles = new Particle[20];
springs = new Spring[19];

for (int i = 0; i < 19; i++) {
  springs[i] = new Spring(particles[i], particles[i+1], 12, 0.4);
}
```

Finally, we will only need to code the movement of the first particle, according to the Lissajus curve:

```
particles[0].setPosition(x1 + cos(angle*a)*radi, y1 + sin(angle*b)*radi);
```

Acknowledgements

I am thankful to the referees that carefully revised the paper for their useful comments and suggestions.

References

[1] B. Fry and C. Reas, *Processing: A Programming Handbook for Visual Designers*, 2nd ed. (2014), The MIT Press.

[2] I. Greenberg, D. Xu, and D. Kumar, *Processing: Creative Coding and Generative Art in Processing 2*, (2013), Friends of ED.

[3] D. Kodicek, *Mathematics and Physics for Programmers*, (2005), Programming Series, Charles River Media.

[4] J. Maeda, ed., *Creative Code*, (2004), Thames & Hudson.

[5] Processing web site. `http://www.processing.org` (as of Feb. 28, 2015).

[6] K. Robinson, *Out of Our Minds. Learning to be Creative*, (2001), Capstone/Wiley.

[7] Z. Ruttkay, *Programming for Artists with Processing*. Proceedings of Bridges 2010: Mathematical Connections in Art, Music, and Science, pp. 555–558, 2010.

[8] D. Shiffman, *The Nature of Code: Simulating Natural Systems with Processing*, (2012), The Magic Book Project, Shannon Fry, ed.