Teaching Mathematics and Physics for Animation in Processing

Lali Barrière

Departament de Matemàtiques, Univ. Politècnica de Catalunya, Catalonia; lali.barriere@upc.edu

Abstract

This paper describes my teaching experience with first year undergraduate students, at the Centre de la Imatge i la Tecnologia Multimèdia (CITM), in Terrassa, a city near Barcelona (Catalonia) during the Fall 2017 semester. The subject I taught was a non-standard curriculum, a mix of mathematics and physics, which I decided to teach using Processing. I called the course Mathematics and Physics for Animation. I have been teaching creative coding since 2008 and many times it served to awake the interest in mathematics of students and artists with no coding education. This time, though, I have used Processing as a tool to teach mathematics (and physics), with a creative approach.

Introduction

Processing [6] is a free and open source programming language and development environment for creative coding in a visual context. Processing works as a Java library, with the implemetation of the primitives and functionalities needed for drawing, animation and interaction, with a very simple, friendly programming environment. This makes it suitable for teaching programming to beginners, especially to audiences with no (or little) technical background. Some teaching experiences with Processing have been carried out in Bridges, as the workshops by Ruttkay [7] and by the author [1]. More in the flavor of the work presented here is the paper by Matsko [5], which decribes an interesting course on mathematics and digital art.

This paper describes my teaching experience with first year undergraduate students, at the Centre de la Imatge i la Tecnologia Multimèdia (CITM) [2] in Terrassa, a city near Barcelona (Catalonia), during the Fall 2017 semester. The subject I taught was a non-standard curriculum, a mix of mathematics and physics, which I decided to teach using Processing. I called the class *Mathematics and Physics for Animation*. Although

Figure 1: *Final project by Saile Wu, a landscape using the* `noise()` *function.*

I have been using Processing for my visual creative projects since 2008 and I have taught workshops and courses to a variety of audiences including undergraduate students from both Engineering and Fine Arts, this is the first time I use Processing as a tool for teaching mathematics and physics. As far as I know, it is the first time that such an experience is carried out at the Universitat Politècnica de Catalunya.

The course is based on the following goals: to think of mathematics and physics from the point of view of animation, realistic or not; to think about challenges and problems of the digital world, but also creative possibilities that knowing mathematics and physics provides in this medium; and, more generally, to acquire a basic knowledge about creative coding as a form of expression. With this aim, I designed a class focused on practical activities, where students could ponder and apply mathematical properties and laws of physics to create graphics and animations. The programming knowledge needed to achieve the course objectives are introduced at the beginning of the course. The approach is mathematical, we use the laws of physics as a tool and we explore the mathematics needed to implement and also modify them.

Mathematics and Physics for Animation outlines a standard syllabus, covering functions and trigonometry, vectors, kinematics and forces, oscillation, with some more ad-hoc subjects as integration methods, physics engines, and color, with a more innovative approach. Among the materials I used to prepare the course, I would like to make a special mention to the book [8] by Shiffman, which comes with examples and covers far more than we needed. I consider it a real jewel. For some technical details I used the book [3] by Kodicek. The course also included the analysis of some artworks by creative coders, and its discussion in classroom. The list of chosen artworks can be found in [4].

In the remaining of the paper I present how the course is designed. After that some examples created by the students illustrate the deployment of the sylabus. Then we briefly comment on the final projects. Figures 1 and 2 show two of them. The paper finishes with a short summary and some concluding remarks.

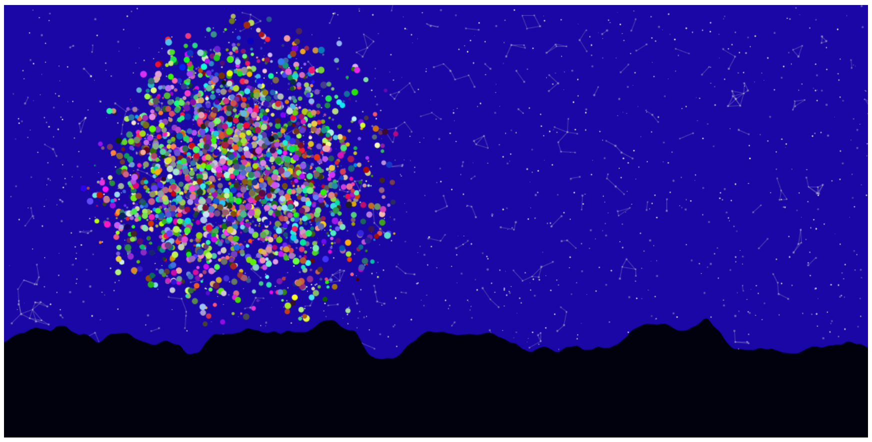


Figure 2: *Final project by Lucía Ríos: a nightscape with constellations and a naive simulation of fireworks.*

Design of the Course

This experience started in June 2017, when I was asked to prepare a course on mathematics and physics to be taught the following Fall (September to December 2017). The CITM is a school of the Universitat Politècnica de Catalunya (UPC) which offers three different bachelor degrees, all of them focused on the demands of the industry and the professional opportunities in the area of Multimedia and Videogames. None of them include a course on Creative Coding.

The students of *Mathematics and Physics for Animation* are undergraduate students from the first semester of the Bachelor Degree in Multimedia coming from mainly two different backgrounds. The majority of them have received scientific and technical preparation, with knowledge in basic mathematics and physics. Some of them, though, come from an artistic education. Only a few have previous programming experience.

Methodology. The course runs over twelve weeks at four hours of class per week, in 24 sessions of two hours. It is built on the following bases:

- Practical work focused on the creative use of (previously studied) mathematics and physics results. The programming knowledge needed to follow the class is, in fact, part of the class: the first sessions need to be dedicated to the basics of programming with Processing. Nevertheless, it is not a programming class, therefore we don't focus on advanced or theoretical programming issues.
- Think of mathematics as a tool for drawing and animating objects. Think of physics from the animation point of view. We don't need to be realistic, but we want to think about the challenges of bringing the laws of physics to the digital world, and the creative possibilities that a deep mathematical knowledge offers to this medium.
- Numerical integration is introduced at a very basic level, to help to understand the lack of accuracy of Euler's integration and the need of a more powerful tool, such as Verlet integration, when simulation needs to be realistic.
- Take into account the creative and aesthetic aspects of your work in the exercises but especially when developing the final project. Think of mathematics as a source of inspiration. Draw your attention to the examples of digital artworks shown in class [4].

The instruction is based on practical work with the support of some previously prepared materials. During the first nine weeks, each session includes a review of previous sessions, a short explanation of the current objectives and an overview of the materials available to achieve the proposed goals which are prepared in the form of guided exercices. This takes 30 to 45 minutes. After that, the students work in pairs to solve the exercices. During the last three weeks, the students work individually on their final project. The instructor is in the classroom, available to comment and help.

Overview of the first year course. Teaching this course presented two main difficulties. First, there were 46 students with only one instructor in the classroom, which is a way too large group for this style of learning. Second, we had to simultaneously teach how to program and how to use mathematics and physics when programming animation. We think that it would be more effective and enjoyable if we could move this course to the second semester, after the students had learned the basics of programming (in Processing or in another language).

In general, the students liked the course. An important part of the group admitted that they were struggling with programming and they found the exercices difficult and sometimes long. At the same time, the world of creative coding was a great discovery for them. The results in terms of academic success were as follows: 41 students passed and only 3 students failed to engage in the course.

Content. *Mathematics and Physics for Animation* covers seven chapters in nine weeks and the development of a final project, partly during the last three weeks and partly as homework. The contents of the seven chapters is the following:

1. **Programming in Processing.** We begin with a short but necessary introduction to Processing.
2. **Mathematics for programming animation: functions, trigonometry and vectors.** We add some mathematics to the basics of programming. We show how functions can be represented in Processing and how they can be interpreted as shapes (in a static graphic) or trajectories (in an animation). We point out that time is measured in frames and length in pixels. Trigonometry and vectors deserve some exercises and the promise of further attention throughout the course. We also introduce randomness as a tool for creation.

3. **Motion in Processing.** We understand motion as a change of position and see different ways of describing it. For instance, using the implicit equations, which are not always possible to integrate, or by adding the velocity to the position at every step. We realize that the latter is a naive application of Euler's method. We work in polar coordinates and see how versatile they are. Examples: a path over a segment, spirals, Lissajous curves; uniform and uniformly accelerated motion (see Figure 3).

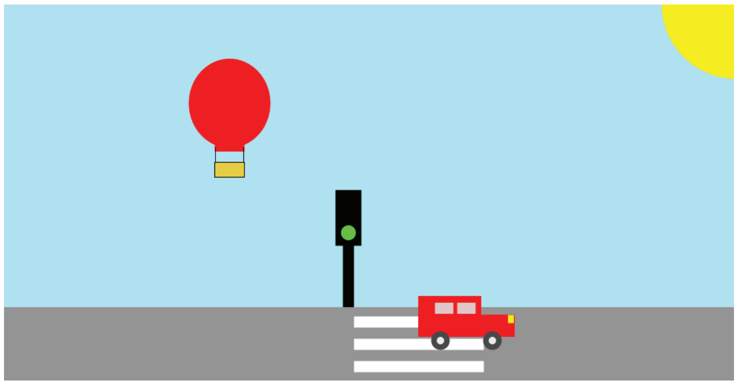


Figure 3: *Moving a car and a ballon in Processing.*

4. **Forces.** Acquire a strategy to apply forces to a moving body, in an object oriented context. Study several kind of forces: gravity and gravitational attraction, friction and resistance, wind, etc. The mathematics behind these forces are not trivially translated to the digital world. For instance, gravitational attraction behaves strangely if the distance between bodies can be smaller than 1 (see Figure 4). Explore the possibilities of realistic forces and create non-realistic forces or forces driven by the interaction. Example: an object accelerates towards the mouse. Which force do we apply? How do we compute it?
5. **Oscillation, waves, springs.** Put trigonometry into action. Oscillation in 1-dimension, with the `sine` function. Translate to code the equation $y(x,t) = \sin(kx + \omega t + \varphi_0)$. Create 2-dimensional oscillators, converting `amplitude`, `angle`, and `angular_velocity` from numbers to vectors (see Figure 5). Apply trigonometry to approximate pendulum motion (see Figure 6). Use vector operations to define elastic forces (see Figure 7).
6. **Rigid bodies and particle systems. Using physics libraries: Box2D, VerletPhysics.** This is a topic that serves as closure of all the physics we worked with by making students realize how difficult the issues that have been left out are, in particular the study of rigid bodies due to the complexity of the physics involved, and the mastering of particle systems because it involves more advanced programming knowledge. Example: simulating smoke with a simple particle system (see Figure 8).

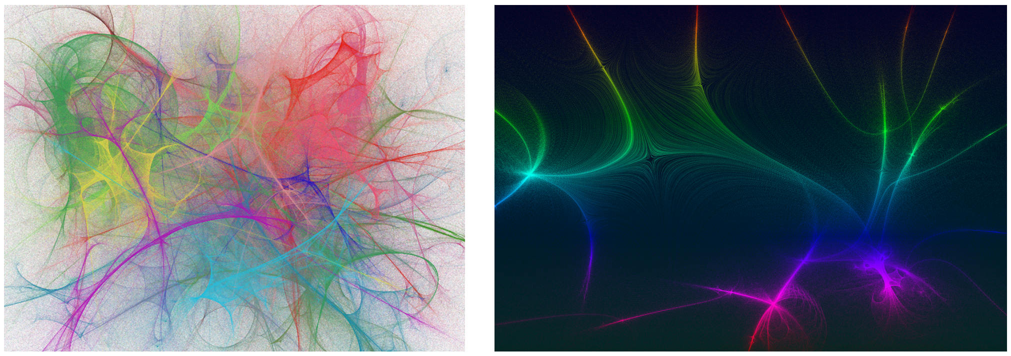


Figure 4: *Particles moving in a field of gravitatory forces, generated by a set of attractors.*

7. **Computational color.** Deal with color as numbers. This topic was intentionally left to the end in an attempt to motivate students to work autonomously and search for information on their own. Since color is appealing and a topic directly related with the aesthetic aspects of the work, many students had already investigated how to use it in Processing. The sessions dedicated to color served to introduce how color is stored and how to work mathematically with it.

In all these topics, as previously stated, rather than emphasizing or focusing on realism or accuracy, we prefer to insist on a new way of thinking about mathematics, using them in a creative manner. For Euler integration, we just want to point out that it is, in fact, what we do, and that this method is not powerful enough when seeking realism. For the use of physics libraries, on the other hand, our aim is to show how some complex problems can be approached working with a simple example.

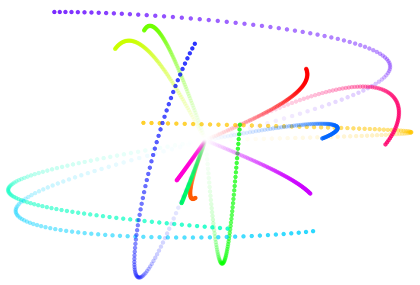


Figure 5: *An array of 2-dimensional oscillators. It uses the HSB mode for colors.*

Topics by Example

In this section we show some of the works the students realized. Due to lack of space, we cannot show all of them, just a selection of the most representative ones.

Motion. The basic strategy for motion, that is, adding the `velocity` to the `position` as shown in Table 1, is one of the fundamentals of the course. Figure 3 shows one frame of an animation of a car and a balloon: the car runs toward the light, waits for the light to be green and restarts, while the ballon follows an oscillatory trajectory.

Table 1: *Basic strategy for motion.*

	One dimension	Two dimensions
No acceleration	`position = position + velocity;`	`position.add(velocity);`
With acceleration	`velocity = velocity + acceleration;` `position = position + velocity;`	`velocity.add(acceleration);` `position.add(velocity);`

Gravitational attraction. This exercise has two intended purposes. First, to explore gravitational attraction, its translation to Processing, and its mathematical possibilities and variations. Second, to work creatively on the aesthetics of the animation and observe how different results can be obtained (see Figure 4).

We want to play with Newton's law of gravitational attraction: $F = G\frac{m_1 \cdot m_2}{d^2}$, but we don't want to be constrained by the accuracy of physics involved. Any modification leading to an appealing result will be accepted. In the implementation of both pictures in Figure 4, thousands of particles are attracted by a small number of attractors. For each single particle, the force acting on it is the sum of the attraction forces from all the attractors. Each force is computed through the following steps:

- Calculate the vector from the particle to the attractor.

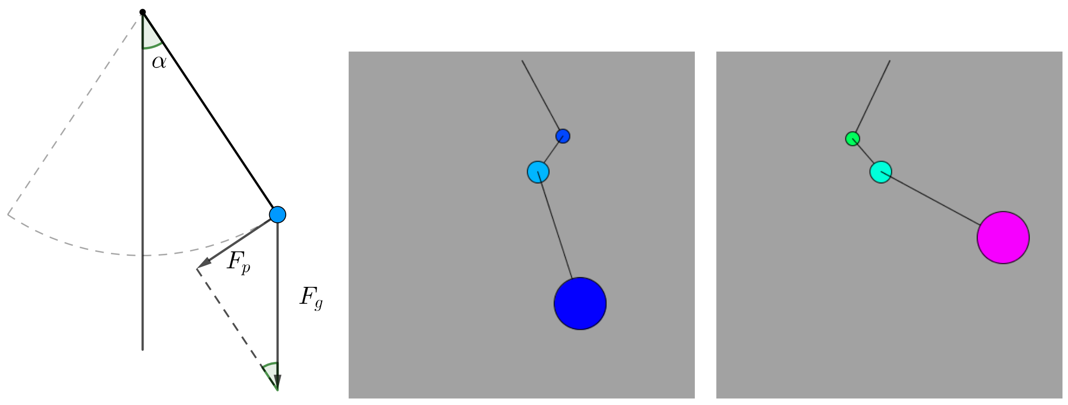
- Calculate the magnitude of this vector, and use it to calculate the strength of the force.
- Normalize the previously computed vector, to have the direction of the force.
- Multiply the direction by the strength.

This is the standard strategy for computing forces and it has been studied for different types of forces before the realization of this exercise.

It is worth noticing that in the examples shown in Figure 4 we are not exactly using Newton's law, but a version without the square in the denominator. Another interesting feature of this animation is that, unlike what we would imagine, particles don't finish their path at one of the attractor's position, because dividing by the distance makes the particles walk away from the attractors when they are too close (i.e. at a distance smaller than 1).

Oscillation. The example in Figure 5 shows an array of 2-dimensional oscillators moving around the center of the frame. Each oscillator has three parameters, `angle`, `angular_velocity` and `amplitude`, all three 2-dimensional vectors. Motion around the center is accomplished by adding the `angular_velocity` to the `angle`. The individual trajectories are, in fact, Lissajous curves, although it is not easy to distinguish them when simultaneously running more than one oscillator, with different parameters.

The pendulum. The equation of the pendulum cannot be analytically solved. Moreover, its numerical integration is far from simple, and not at all at the level that our students can understand.



(a) **(b)** **(c)**

Figure 6: *(a) Studying the pendulum. (b) and (c) Trying a naive double pendulum.*

According to the pendulum equation, the angular acceleration depends on the gravity, the angle (α in Figure 6), and the length of the pendulum's arm. It easily, if not accurately, translates to code:

```
angular_acceleration = (-1 * gravity / len) * sin(angle);
```

After this computation, the angular acceleration is added to the angular velocity, which in turn is added to the angle. The use of this approximation opens the door to designs like arrays of pendulums running in parallel, or a (naive) simulation of a double pendulum, as shown in Figure 6, with room for interaction.

Springs. The trick in the examples from Figure 7 is to build a string of springs and make them move in one way or another. The difficulty here is that we don't only have a moving object, a particle, but a set of objects, the particles, with interactions between them, the springs. Each spring connects two particles, say particle A and particle B, and for each spring we need to compute two forces, one for each particle. For each spring, the corresponding forces are computed through the following steps:

- Calculate the vector from particle A to particle B.

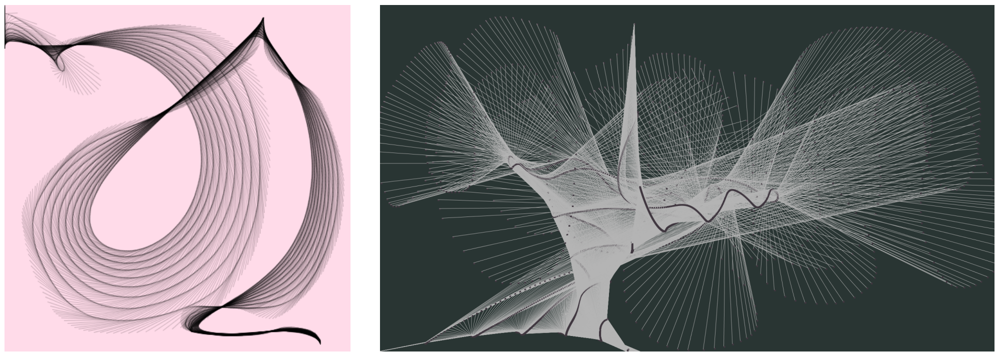


Figure 7: *Experimenting with a chain of springs, with different levels of control.*

- Calculate the magnitude of this vector, and use it to calculate the strength of the force, according to Hooke's law: $F = -k(\ell - d)$, where ℓ is the length of the vector and d the length of the spring at rest.
- Normalize the previously computed vector, to have the direction of the force.
- Multiply the direction by the strength and apply the force to particle B, then multiply it by -1 to apply it to particle A.

Of course we have to pay attention to the direction of the force, which has to be opposite to the stretching of the spring.

Particle systems. We only did one example of particle system. One of its realizations is shown in Figure 8. As expected, almost all the students needed help, because the programming difficulties involved:

- As we learned along the course, a particle has its own position, velocity and acceleration, its drawing method and the method to apply a force.
- We work with a list of particles that need to be dynamically created and destroyed. We cannot make do with `arrays`, we need an `ArrayList` to be able to perform insertion and deletion.
- The particle system is an OOP class in itself, works as an engine, which for students that have just learned how to program is quite an advanced concept.

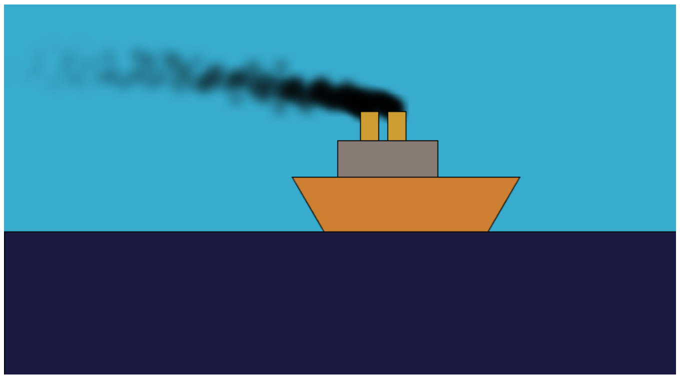


Figure 8: *A simple particle system for the smoke.*

Final Projects

In the final project students were asked to use what they learned in a creative animation or still. We focus on being creative, more than an exhaustive use of the techniques covered during the first part of the course. Two examples illustrate what students were able to accomplish:

- Figure 1 comes from the final project by Saile Wu, who decided to create a landscape by using the `noise()` function. He uses also pure randomness, and makes a nice use of color.
- Lucía Ríos created a nightscape with an interaction that triggers fireworks. The simulation of the fireworks is quite simple but aesthetically effective. A frame of this animation is shown in Figure 2.

I was surprised by the fact that very few students were interested in abstraction, although we had seen and discussed many abstract digital artworks in class. The final projects were almost all figurative, mainly landscapes and simulations, with or without animation. This made a striking contrast with my experience in Fine Arts schools, where students have a clear interest in geometry and abstraction.

Summary and Conclusions

I have presented here the course *Mathematics and Physics for Animation*, an experience with undergraduate students using Processing to teach mathematics and physics. The design of the course takes into account the following aspects:

- The course has to begin with an introduction to programming.
- The syllabus is long and covers a variety of topics from mathematics and physics. Although most of them were already known, the approach is new.
- Mathematics and physics are taught from an applied and creative point of view. With this purpose, it is important to integrate the aesthetic ideas into the work, but also to discuss digital artworks that use creative coding in class.

I have thoroughly enjoyed this experience and I know most of the students did as well. I'm greateful to all of them, especially because this was a big challenge: not only did most of them face programming for the first time, but they encountered a new way of using mathematics and a new way to understand physics. It was my intention to make the course fun and engaging. It wouldn't have been possible without them doing their part of the job.

References

[1] L. Barrière. "Lissajous Curves: an Experiment in Creative Coding." *Bridges Conference Proceedings*, Baltimore, MA, Jul. 29–Aug. 1, 2015, pp. 549–544.
http://archive.bridgesmathart.org/2015/bridges2015-549.html

[2] Centre de la Imatge i la Tecnologia Multimèdia (CITM) website. http://citm.upc.edu/ (as of January 15, 2017).

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

[4] Links to artworks using creative coding, mathematics and physics.
http://mat-web.upc.edu/people/lali.barriere/mfa-links.html (as of January 15, 2017).

[5] V. Makso. "A Mathematics and Digital Art Course." *Bridges Conference Proceedings*, Waterloo, Ontario, Canada, Jul. 27–31, 2017, pp. 261–268.
http://archive.bridgesmathart.org/2017/bridges2017-261.html

[6] Processing web site. http://www.processing.org (as of January 15, 2017).

[7] Z. Ruttkay. "Programming for Artists with Processing." *Bridges Conference Proceedings*, Pecs, Hungary, Jul. 24–28, 2010, pp. 555–558.
http://archive.bridgesmathart.org/2010/bridges2010-555.html

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