Zixuan Kevin Fan
N00631555
09/11/17
Major Studio I

When I look back at all the 5 projects I made last week. An recurring thread keeps poping out in my mind, science! Planetary systems, fireworks, repelling balls, terrain simulation, and galaxy etc. They all require some degree of understandings of physics and math in order to compose. I'm extremely grateful of my previous educations of science and engineering during my undergraduate studies. Yet, isn't my background somewhat dictates what I'm capable of and actually doing right now?

The nature of code
When we think about animations, someone said best that we're not creating anything original, but essentially are inspired by and pirating the inventions of nature. If we examine into several fundamental movements of any types of animations, translation, scaling, acceleration, and repellency. We can find their origins in nature. Translation happens because of the linear movement of the objects; scaling happens when the density of objects change; acceleration occurs largely due to the earth's gravitational pull; and repellency is triggered according to the momentum of two colliding objects.

After we realize that almost all animations are rooted deeply in the science of physics and math, it is much easier to conjure up the coding to make it happen. Our research in natural science can cover all the background knowledge it requires to make any movement happen. The only task that is left for us to do is to compose the exact combination and timing of the each individual animation to produce the effect we want.

Science for artists
Artists/designers I talked to really have a love-hate relationship with science. Large part of them chose to pursue their career in arts given that they recognize their lack of passion in studying science and engineering. However, with the arms of science and engineering, it is way easier to think of how to realize certain movements and compose the resulting keyframes.

Computer graphics started off as a branch of computer science. The term itself was coined in the 1960s by researchers Verne Hudson and William Fetter of Boeing, a aerospace company. In fact, all the softwares we use like Adobe Creative Suites, Processing, openFrameworks or 3D softwares such as Rhino and cinema4D, compile the graphic input of artists to high-level code, then machine code and binary files. More and more design softwares nowadays also encourage people to think using code to design and prototype: you can use *expression* in After Effect to pinpoint animations or in Framer.js, you have to write Vanilla.js (a variation of Javascript) to prototype. By using any design softwares with graphic interface, it actually adds another layer of abstraction between humans and computers. Plus, the learning curves for using such softwares are usually steep and you have to remember all the keyboard shortcuts in order to speed up your work. Wouldn't it be easier for artists to design everything in code or at least a high-level language so that they will never have to worry about version compatibility and learning new interfaces? It is a constant debate in the field of design and engineering, the tradeoff between complexity and usability.

Creative coding is not programming

For this part of the discussion is not trying to intimidate anyone who’s interested learning code or is already learning creative coding, nor criticize the work of artists who code in this very way in the industry. The purpose of this section is to recognize the differences between creative coding and profession programming in the software industry. Hopefully two fields of studies can learn from each other and respect the process of each other’s work.

Creative coding is all about “making things work”, regardless of the use of any plugins or preset packages, as long as we produce the results we want. In this process of realization, we are often stuck in the confusion of picking up new tools and choosing the right add-ons. And a lot of times, to be honest, we don’t really know what happens when the code just magically compiles.

In the industry of software engineering, reliability, optimization, readability and modularity are the priorities when we design any architecture or programs. In creative coding, we can afford to have errors at times when an animation happens; however, in software engineering, it is the first rubric people judge the code upon. In creative coding, the computer can be slow and we can use up all the memories and CPU when the program runs, whereas in programming, people are trying to cut down every single percentage of the CPU and every bit of the RAM. When in creative coding, we are often by ourselves write code less than a thousand lines, in software industry, it is a collaborative process often among more than ten people to work on the same project. Readability and documentation can be extremely important in such context. Most of the times, artists would put all the code in a single file, while in software engineering, people would break it down according to their functionalities and methods for quick update and corrections.

I believe each field has to learn from each other, as creative coders and artists can get inspirations from computer science and produce work that is both aesthetically attractive and efficient. Software engineers can also discover the underlying design concept and how artists utilize simple tools to imagine complex artwork. The 5 in 5 project is definitely rewarding and I already decided to make it a 30 days project to further extend my knowledge in computer animations and motion graphics.

References

1. Computer graphics