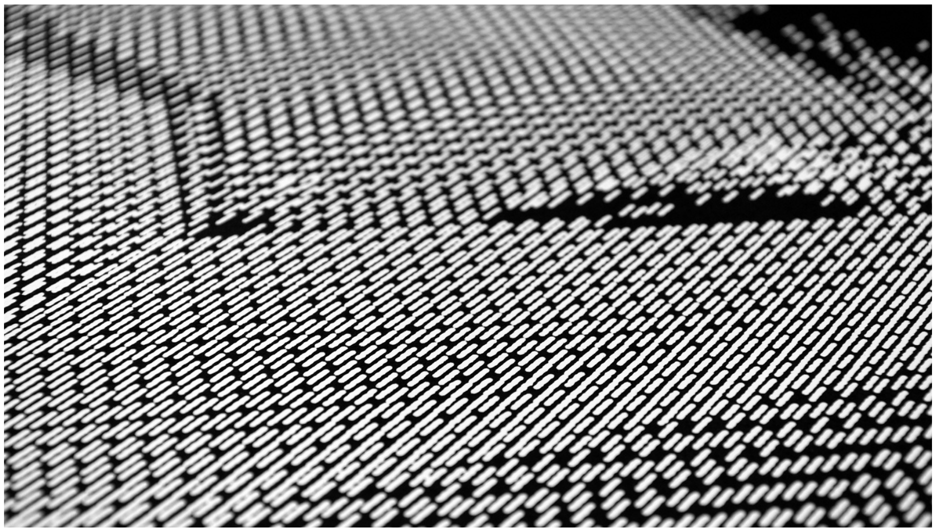
Processing Foundation Follow
The Processing Foundation promotes software literacy within the visual arts, and visual literac…
Feb 1 · 10 min read

Thoughts on Software for the Visual Arts

by Casey Reas


Detail of HSB-119–006–090–1366–618 / HSB-135–006–090–1232–687, 2015.

I make things—all the time. In the studio, I use tools every day. Some of the tools are "hard" like a screwdriver but most of them are "soft." I move back and forth between making software tools, using my tools, and using tools created by others.

I need to have precise control of my tools to form my ideas and I need to be able to modify my tools to explore new ideas. I want to believe that I can form unique ideas, in contrast to accepting the ideas that are encoded into the software tools that I'm using. Through the tools that I make, modify, and use, I sometimes feel like I am in control and sometimes I feel like my ideas are heavily biased by software that others have made. This is the quandary that I'm thinking about today.

Idea

I have ideas about how software tools can be improved for myself and communities of other creators. I want to be a part of creating a future that I have experienced only in fits and starts in the recent past and present. I have seen independent creators build local and networked communities to share intellectual resources and tools. The individuals in these communities share the responsibility to contribute ideas and infrastructure to making their own tools. It's an aspiration toward a way of making and sharing that has been strongest in one area of the visual arts, the world of creator-programmers. I want to try to scale it within that context and I also want to know if it's applicable to other areas.

The converse to this is more widespread. In this model, creators pay software companies to license the tools that the software company has defined and produced. The essence is paying a company to make decisions about software tools in exchange for not having to conceive one's own tools.

Both of the models described above break down in wide practice and both are ideals from different points of view. One privileges flexibility, freedom, and collective responsibility and the other promises ease of use in exchange for payment. Because neither model works well in the present, I suggest that we should work toward a model that empowers creators to control their own tools.

This isn't an argument for one economic system over another, it's about shared infrastructure for creative work with a collective goal to create flexible tools. It's about empowering people to create through access to tools and platforms. This is also not an argument about dissolving intellectual property and copyrights; it's about finding a balance between shared common infrastructure and individual ownership.

This aspiration is possible through free software where the word free relates to "freedom", not "free lunch"—because we all know there is no such thing. Along with free software, we also need open standards and collective desire to be in control of our platforms and tools.

The contrast to this model is proprietary knowledge and resources. Today, creative communities rely almost exclusively on for-profit corporations to create and control their tools. Companies package what could be modular, general systems into a monolithic, proprietary products. The result is that it's easy to color correct a photograph if you have the financial resources for the product or you are willing to break the law by using a cracked copy, but creative control is stifled when an idea is outside the boundaries of a product that can't be extended or adapted.

So far, this short text has been general and vague, but I want to be specific and clear. As one case study, I will discuss the development of the Processing software that I co-founded in 2001 with Ben Fry. Processing is one example of a free, open, and modular software system with a focus on collaboration and community. It's the example that I know intimately inside and out so I will use it to reflect on the ideas sketched out in the first part of this text.

Platform

The Processing software is an integrated programming language and environment. It's primarily for students and professionals within the visual arts including design, art, and architecture, but it has evolved in time to find a place within the humanities and sciences, even including university computer science programs and high school math and science classes.

From the start, Processing was created as free and open-source software (FOSS) to be accessible and flexible. By access, we mean two things. First, that people can get it; it can be downloaded without cost —it's "free." Second, we mean that it can be understood by a general audience. Processing is simple, but not simplified. We aspire to make the interface easy to use and the documentation clear and free of unnecessary technical jargon.

Ben and I grew up with the first generation of home computers and the culture surrounding them permeated our environments. Computers were simpler then and the code was too. Everything felt possible and we were ready.The hardware and software was designed to be modified and diagrams and code for both were often shared. It was expected that a computer user was also a computer programmer—how else would you make the machine do what you wanted it to? If you liked games, you could write your own. If you liked music, you could write a program to help you compose. The computer was an environment for creation and authorship. To reference Howard Rheingold, computers were Tools for Thought.

About fifteen years later when we were in our twenties, we experienced and participated in the initial spread of the world wide web. The web extended values from decades prior—it accelerated the promise of more universal access to information, of creating new kinds of communities, and of breaking down hierarchies. These values are shared with the origins of Processing.

As an example, in the first years of the web, many people learned to create web pages by reading sites' HTML directly through the "View Source" feature built into browsers. Inspired by this open quality, the early versions of Processing had an export feature that, by default, included the source code as well as a web-ready files that could be uploaded to a server to share the work with an international audience.

Processing was positioned in a unique space when it was first launched in 2001. It wasn't a tool to make programming easy for visual artists in the tradition of Hypercard and Director. It also wasn't a programming language for professionals in the tradition of C++ and Java. It was a place between, a middle ground where visual artists and designers could be confident with their ability to work with form and images while learning programming and engineers could be confident in their ability to write code while learning about form and images.

It's hard to pin down what Processing is, precisely. I admit, it can be confusing, but here it is: it's both a programming environment and a programming language, but it's also an approach to building a software tool that incorporates its community into the definition. It's more accurate to call Processing a platform—a platform for experimentation, thinking, and learning. It's a foundation and beginning more than a conclusion.

Processing was (and still is) made for sketching and it was created as a space for collaboration. It was born at the MIT Media Lab, a place where C. P. Snow's two cultures (the humanities and the sciences) could synthesize. Processing had the idea to expand this synthesis out of the Lab and into new communities with a focus on access, distribution, and community. Processing is what it is today because of the initial decisions that Ben and I made back in 2001 and the subsequent ways we've listened to the community and incorporated contributions and feedback since the beginning. Processing was inspired by the programming languages BASIC and Logo in general, and specifically by John Maeda's Design By Numbers, C++ code created by the Visual Language Workshop and Aesthetics and Computation Group at the MIT Media Lab, and PostScript. Processing wasn't pulled from the air, it was deeply rooted in decades of prior work.

Process

While Processing started out as the work of two people who volunteered their time, it quickly outgrew what was possible for Ben and I to manage. Increased expectations and ambitions for Processing emerged when other people started to use it. Early on, we needed to do two things. First, to figure out how to collaborate with other people and second, how to remove ourselves as bottlenecks to moving forward.

For collaboration, we found amazing people through the internet who were excited to volunteer time to work with us on certain aspects of the project. However, we weren't able to find help for some of the more technically difficult under-the-hood programming tasks. So from early on, we've needed to secure some funding for the project to hire out some pieces of the code in balance with working closely with volunteers. As some collaborators contributed more to the work, they organically moved closer to the center and into more specific roles within the project. Over time, Florian Jenett, Andreas Schlegel, Elie Zananiri, Andres Colubri, Dan Shiffman, and Scott Murray became essential. Dan became a third official project lead when we started the Processing Foundation in 2012. Many, many other volunteers made crucial contributions over the years—too many to list here, but it's all archived at processing.org/people.

In addition, Ben and I were spending our evenings and weekends working on the project and that has largely continued into the present, but we now require more balance. Like our collaborators, we now have even more challenging responsibilities outside of working on Processing. That brings us to the second point, the need to remove the core team as bottlenecks to the growth of the project.

This is done through more shared responsibility and developing the software in a modular way. The goal for Processing has always been to have a minimal code base and interface. It's a different type of software development than a program that is sold and marketed based on new features that are continuously added and removed to encourage or force upgrades. Processing has a core that changes slowly, while the structure of the code supports libraries to extend the software quickly into new areas.

A Processing library is a standalone piece of code that integrates into the core to extend what is possible. With only a few exceptions, libraries are contributed by the community of people who use Processing. The generous developers who make and share libraries document their open code as well as host the files for download. More than anything, libraries have allowed Processing to expand into unexpected directions and they are a remarkable example of a community of individuals sharing responsibility for building and maintaining a free software infrastructure.

As a free software project, Processing utilizes other free software projects. Processing was built by combining modular pieces of free software together and adding more code to create a new coherent whole. If the entire project was written from scratch, it would have required a team of engineers and more time. We had neither.

It's also important to say that we didn't want to raise money to write Processing from scratch and we didn't want to work on Processing full time or to manage a team of people to work on Processing full time. We made Processing to help us with our primary work. In Ben's case, this was creating visualizations for the Human Genome Project and in my case, to teach designers the basics of computer programing and to explore code in my visual arts practice. We needed a tool to support the work we did—to develop ideas and forms in our own context. We had (and still have) no interest in working full time to make a tool.

Our system of guidelines and relationships that enabled the software to be maintained and to improve broke down slowly and reached a critical point around the time of the Processing 2.0 release. The expectations of the community and the complexity of the software had grown to a point where volunteered "free time" of the core developers and occasional help could not complete the work without deep personal sacrifices. To attempt to keep the project moving, we started the

Processing Foundation as a legal not-for-profit 501(c)(3) organization. We started to ask for donations from the community at the time the software is downloaded. The truth is that we needed substantial funding to keep the software moving forward and the ideal of a 100% volunteer effort coordinated through the internet wasn't working for our specific situation. We finally acknowledged that free software is expensive to make.

Processing evolved through building on top of existing tools and collaborating with others to share the responsibility. This is still the case today, but the development is also supplemented through donations from the community, through programs like Google's Summer of Code, and occasional generosity from academic institutions (New York University, Miami University, University of Denver), companies (O'Reilly), and other open-source projects that use our code (Arduino).

On the tenth anniversary of the Processing software in 2011, we made a list of what we felt was essential to the project:

- Programming in an arts context
- Simple but not simplified, scale complexity
- Made for education and learning
- Bridge to other languages and platforms
- Provide infrastructure for learning and teaching
- Develop through teaching
- Simple publishing for sharing
- Community infrastructure
- Extensible through libraries
- Import/export to diverse media and formats

In 2015, I re-assessed this list and synthesized it to this core:

- Access
- Community

- Free (Libre, Libero)

I feel that with more detail (as I have started to flesh out above), these three points are the core of Processing and they differentiate its approach from proprietary, consumer-driven software.

Domain

These core ideas outlined in this text emerged within the culture of free software. Free software is primarily created by technical folks for other technical folks—it has been most successful in the realm of systems administration and operating systems in projects like the Apache Web Server and GNU/Linux. The Free Software Foundation and its "copyleft" idea has been the pioneer and uncompromising proponent that "any user can study the source code, modify it, and share the program."

Artists have also pioneered new ideas about intellectual property. For instance, the first Radical Software publication in 1970 introduced an anti-copyright symbol, an "x" within a circle to mean "DO copy." Dan Sandin introduced his Distribution Religion in the early 1970s so the schematics for his Image Processor could be "copied by individuals and not-for-profit institutions without charge."

Through initiatives like Processing, communities of creators are working to realize a new vision for software within the arts with the goal of controlling our own tools. In time, will this grow or diminish? Is this a trend or is it more substantial? The model of our communities paying a company for licenses to use standardized software that will "just work" is a model that might make sense in the category of functional productivity software, but has little relevance to artists and designers who thrive on radical exploration. I want to succeed in pursuing this new path; I want you to succeed; I think it's important and it can be done.