Zs. Ruttkay, Programming for Artists, Proceedings of Bridges 2010: Mathematics, Music, Art, Architecture, Culture, Pécs, Hungary, 24–28 July 2010. pp. 555-558.

Programming for Artists with Processing

Zsófia Ruttkay
University of Twente, Dept. of Computer Science, Creative Technology
P.O. Box 217
7500 AE Enschede, The Netherlands
E-mail: z.m.ruttkay@ewi.utwente.nl

Abstract

Programming has not yet become a natural medium for artists. I see two, connected causes for this: to acquire sufficient skill of computer programming and the essential algorithmic thinking is normally beyond the capacity of artists, and, for many, it is not apparent what novel possibilities the computer offers for visual exploration and for entirely new genres of experience. Processing – a recent open-source programming language made by artists for artists – makes it feasible to break both of these forbidding arguments. The goal of the workshop is to let participants – including artists who have never ever programmed before – get an insight into the programming language and its environment, practice making visual sketches and control them by using the mouse, and get an outlook on the potentials by browsing selected digital artworks. The workshop will be organized around a series of algorithmic visual tasks, to illustrate the basic algorithmic concepts and language constructs. By the end of the workshop the participants will have created some own digital sketches, and will have a basis to develop their programming knowledge further. This outcome may enrich the arsenal of individual artists. It is also useful for educators who seek a first, attractive programming language to be taught in the secondary school, or for non computer science students.

1. Introduction

Programming has not yet become a natural medium for artists. Looking at research papers and artworks where computer was used at earlier Bridges conferences, we hardly find such single-author artist mastering programming who had no prior mathematics or engineering education before or parallel to his/her art studies. In my opinion there are two causes for this situation:

- to acquire sufficient skill in 'serious' computer programming languages such as Java or C++, and the essential algorithmic thinking is normally beyond the capacity of artists,
- for many artists, it is not apparent what novel means the computer offers for visual exploration and, even more, for entirely new genres of experience.

Let's look at the two causes, which are indeed interrelated. While there have been 'easy to learn' programming languages around (like the famous Logo [1], or DBN, Design By Numbers [2]), these by nature offer a limited range of graphical output, as of complexity and aesthetical quality. They are good to get the taste of algorithmic thinking, but for serious works, one has to switch to one of the 'real' programming languages. These are far too different, more complex and harder to use. Moreover, these languages were not developed especially for creating graphical output, but for all kinds of professional programming tasks. The beginner has to fight through a lot of technical and non-inspiring aspects of the language till manages to create something visible on the screen.

In another category are the dedicated tools, e.g. to make tessellations of fractal art. With these the artists can produce only the certain genre of graphical output the tools was made for (like Julia and Mandelbrot

sets). Moreover, it requires long experimentation and (trial and error) learning to find one's way to some original and satisfactory outputs. Unless one has from somewhere else the insight into the underlying mathematical phenomena, the usage of parameters and tweaking the sw remains a matter of exploration with surprises, not a design-like process.

Thus no wonder that real programming is considered to be out of reach for artists, and that they are disappointed by the limitations of the tools they are able to use. Hence, it is hardly experienced that a free, general-purpose 3d modeling or programming environment may give wings to creativity, in several ways:

- creating virtual models is usually cheaper, faster and easier than making physical prototypes – thus the digital model allows an efficient exploration and design for some tangible final piece;
- if an abstract mathematical representation and parametric framework is set up for making a digital model, it is possible to generate a lot of variants, also ones which were beyond the artists' mental imagination;
- last but not least: digital artifacts can change in time, opening the entirely new domain of animation and interactive experience. Here it is the *code* which itself becomes the material of the artist, and – just as when working with traditional tangible medium – challenges him to use it for something new, surprising, never seen before. And heard before, as the visual output can be combined with sound, or even with triggers for other senses like vibration. On the control side of interaction, it may be the conscious act of a single person, or the overall unconscious movement of by-passers in a public space, or urban traffic, or changes in environmental data which are the ultimate input parameters for some artistic visual or multimodal effects. Thus programming is an unique medium per se.

Processing – a recently developed open-source programming language "made by artists for artists" – makes it possible to break through these forbidding arguments, and offers the possibility for artists after a friendly and relatively short learning period to try out their hands in the above domains . The main goal of the workshop is to make novices familiar with the possibilities of Processing, as a tool which they *can* learn and use for their own purposes.

During the workshop participants – including artists who have never ever programmed before – will get insight into the programming language and its environment, will practice making visual sketches and how to control them using light, sound or gesture, and get an outlook on the potentials by reviewing selected digital artworks. It is impossible to learn any programming language in such a short time. But I expect that participants' prior concerns of programming will be washed off by intrigue and interest in algorithmic art, and in using Processing for the making. As of the practical aspects, having seen the basics and the rich learning resources of Processing, participants will leave the workshop with the confidence that they can develop their programming knowledge further.

This outcome may enrich the arsenal of individual artists. It is also useful for educators who seek a first, attractive programming language to be taught in the secondary school, or for non computer science students.

2. The Processing Language

Processing grew out of the initiatives at MIT Aesthetics and Computation group led by John Maeda between 1996 and 2003 [3]. The first public version of processing was published in Fall 2002, just after a year Ben Fry and Casey Reas started development [4]. It got warm response and grew into an open project, enriched by feedback, fascinating examples of usage and community-created libraries to share.

Today the official web site [5] offers the (free) 1.0.X alpha versions for GNU/Linux, Mac OS X, and Windows. Unlike its predecessor, the above-mentioned DBN, it is a fully expressive programming language. For advanced users, it is possible to use all the expressive power of the Java language. But for the novice programmer, it is a language with a simple syntax and with a reasonably small set of language vocabulary – which can be naturally extended with experience.

Processing was designed to make interactive graphics. Without writing a single instruction, a drawing canvas appears. This canvas, by using dedicated language elements and a flexible color model, can be populated (by writing text-based code) by 2d pixel-based or vector-based images and 3d models, which can be explored using the built-in navigation and lighting engine. It tells much about the intended usage of the language that programs are called 'sketches'. The language allows making experiments, investigations in a few lines of code. From these initial small sketches more elaborate ones may be created, by re-using, extending or modifying them in an incremental way.

Besides the attractive graphical features, the language offers easy to use mouse- and keyboard-based interaction. Beyond that, it allows network communication, control via camera and image processing, and there are ready to use libraries for sound input and output, as well as for electronic microcontrollers. Thus it can be used for a range of applications from creating artistic drawings, via driving interactive installations or visual and acoustic effects in theaters to data visualization. Moreover, as Processing is a simplified Java language, with all the essential constructs of it albeit in a simplifing 'packing', after having learnt Processing it is easy to switch to other general languages like Java or C++, or ActionScript for web applications or OpenGL for sophisticated computer graphics.

3. The Learning Environment

Processing is integrated in a rich yet friendly development environment, to facilitate a smooth and incremental learning process for individuals. The major features, in a nutshell:

- the environment integrates the text editor for writing the code, the entire language reference in a basic and a full set, and a single-button-click compiler to see the result;
- the language itself can be used on three levels:
    - single-line commands can be typed in to try out shapes;
    - procedures and objects can be defined (and re-used, also from dedicated libraries) to make more complex, interactive behavior;
    - advanced users may include usual Java code in Processing.
- the environment locally offers example sketches to illustrate the usage of language constructs as well as special tasks and the usage of some libraries;
- the on-line repository offers especially made for novices, but conceptually sound learning resources (e.g. tutorials);
- the by the day growing exhibition from submissions from the community is a feast and inspiration. As in most cases the code is also included, a valuable insight on 'how it was made'.

Finally, the low-key wordings but highly professional and ready to use content, trust-based sharing of code and knowledge on the related web-sites, and above all, creativity and aesthetics as ultimate criteria for community-based feedback make also the open philosophy of Processing unique among other programming languages.

Zs. Ruttkay, Programming for Artists, Proceedings of Bridges 2010: Mathematics, Music, Art, Architecture, Culture, Pécs, Hungary, 24–28 July 2010. pp. 555-558.

4. Organization of the Workshop

The workshop will start with a short introduction about the nature of the Processing language and a showcase of works (sketches) made with it, followed by a demonstration of how to use the processing programming and learning environment. Then in the rest of the workshop participants will be challenged by a series of visual tasks, one after the other. Participants will be working on their own (may be in pairs), and assisted individually. Some sketches will be looked at and discussed jointly.

Taking a visual task as a starting point has several benefits:

- it is familiar and motivating to artists;
- as the tasks are all somewhat open, not fully defined, it gives space to creativity and a variety of interesting visual outputs;
- from a hand-drawn sketch an algorithmic production plan will be created in natural language, which will give a natural context for introducing the language constructs necessary to program the intention in Processing;
- by making a series of yet more complex variants of a visual task, these can be dealt with by re-use and extension of previous pieces of code.

Below we give an illustration of such tasks, with the mathematical and algorithmic concepts and the outlook to further variants and language constructs. Some sketches for the tasks may be seen on-line [6].

	The visual task	**Algorithmic concepts**	**Outlook**
1.	make a composition of some n lines in a regular way	variables, parameters, repetition, loop, array, line	Other geometric primitives Color
2.	make a composition as above which changes with time	(illusion of) motion animation	kinematic and dynamic motion
3.	make a composition as above which reacts to the mouse	Interaction, event, input/output	light, motion, and sound events as means of control
4.	make an irregular composition of n lines	Randomness, generating random numbers	Perlin noise

As of technical needs: each participant has to have own laptop or PC (Mac or Windows). Internet access would be good, but not a must.

References

[1] The LOGO Foundation http://el.media.mit.edu/Logo-foundation/logo/
[2] Design by Numbers, http://dbn.media.mit.edu/
[3] MIT Aesthetics and Computation Group http://acg.media.mit.edu/
[4] Casey Reas and Ben Fry, *Processing: A Programming Handbook for Visual Designers and Artists*, MIT Press, 2007.
[4] Ira Greenberg, Processing: Creative Coding and Computational Art (Foundation), Friends of Ed. 2007.
[5] The Processing web site http://www.pocessing.org
[6] http://create.mome.hu/ruttkay/visualtasks/

The above links were visited on 22.04.2010.