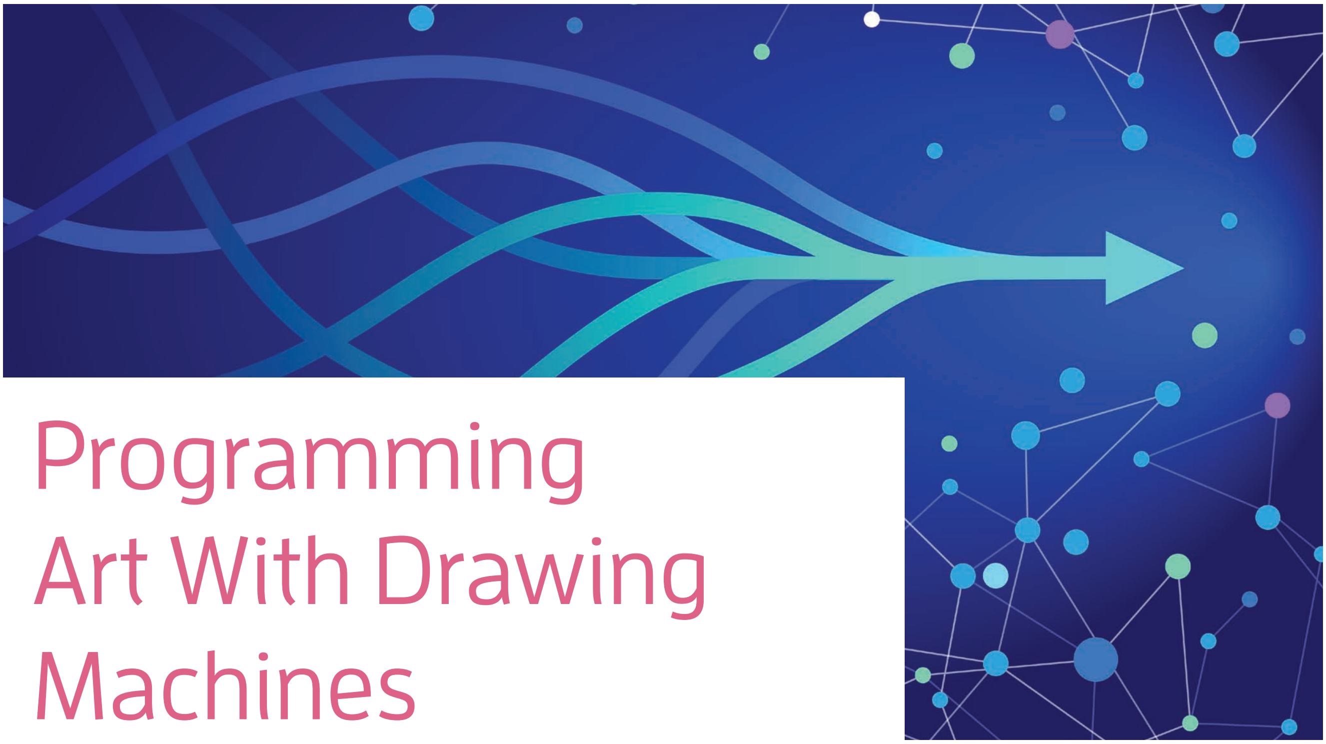


COPYRIGHT ISTOCKPHOTO, CREDIT:LVCANDY

Programming Art With Drawing Machines

Benoit Baudry, Université de Montréal and KTH Royal Institute of Technology

Martin Monperrus, KTH Royal Institute of Technology

Algorithmic artists master programming to create art. Specialized libraries and hardware devices such as pen plotters support their practice.

Algorithmic artists are those who use code and programming as their medium of choice for artistic expression.[1] They create images, films, sculpture, and music with code. When those creative algorithms make use of randomness, this is called "generative art." There is a fundamental difference between enterprise coding and creative coding. For enterprise applications, a computer monitor suffices. For creative coding, there is an intense desire for physicalization, for transmuting the digital code of the art piece into a tangible object in the physical world. In this column, we give an introduction of algorithmic artistry and deep dive into the favorite physicalization machine: the pen plotter (Figure 1).

ALGORITHMIC ART PRIMEUR

Algorithmic artists write code. In Figure 2 we illustrate how code can let artists develop a cohesive practice across the digital and the physical worlds. In Figure 2(a) we show an excerpt of a program written by an artist; it produces the digital version of the artwork. This program is implemented in Javascript with the p5.js library,[a] which powers millions of digital artworks on platforms such as fxhash[b] or feral files.[c] This function draws five different parts of the artwork, in a circular shape. The different random choices indicate that the artist has let the randomness decide the rendering of some parts of the artwork. This makes the artwork generative by nature: every time the program executes, it produces a piece that is unique, and yet, all pieces are part of a cohesive series as they all follow the same algorithm and

Digital Object Identifier 10.1109/MC.2024.3385049
Date of current version: 26 June 2024

[a]https://p5js.org/.
[b]https://www.fxhash.xyz/.
[c]https://feralfile.com/.

EDITOR PHIL LAPLANTE
IEEE Fellow;
plaplante@psu.edu

the same structure. This program allows the art to exist in the digital world, rendered on some display. For example, collectors can see it on their laptops, phones, or a wall screen in a gallery.

Some artists wish to distribute the same generative artwork in the physical world. For that they need another form of rendering of the artwork, possibly a machine that will physically draw the piece in real space-time. Such a drawing machine, holding a real pen, is called a pen plotter.

PHYSICALIZATION OF EXECUTION

The physicalization of execution is as old as programming. In fact, before the invention of the computer monitor, rendering the execution of a generative procedure had to involve a physical procedure. In the early 19th century, Jacquard designed and created the Jacquard loom,[2] one of the oldest programmable machines, meant to draw patterns. These mechanical machines were controlled by punched cards (the input program) in order to weave complex patterns into textile (the output). Fast forwarding to the 1950s, Desmond Paul Henry repurposed a bombsight computer from World War II into a drawing machine that leveraged the computer's internal moving parts.[3]

Through the 1950s and 1960s, the execution of early computer programs was physicalized through typewriters or pen plotters[4] (Figure 3). A pen plotter is a machine on which a pen is fixed and that can be controlled to move on a 2D space, to draw on a sheet of paper. In these days, drawing machines were essential for designers, architects, and artists because monitors were too small or too primitive. For example, the visualizations of early digital animations were created by plotting on a microfilm.[5]

While the pen plotter industry was active in the 1960s and onward, virtually all pen plotter manufacturers disappeared when laser printers became cheap and precise enough in the 1990s. However, pen plotters and drawing machines have enjoyed a renaissance since the 2010s, entirely driven by

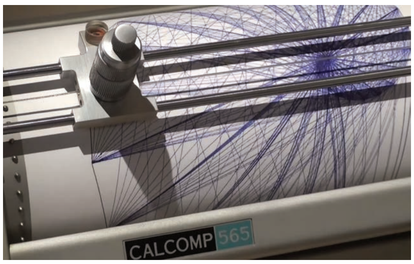
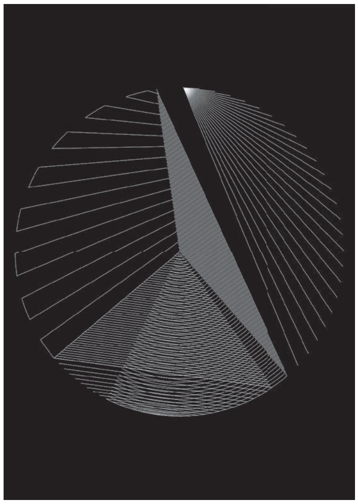


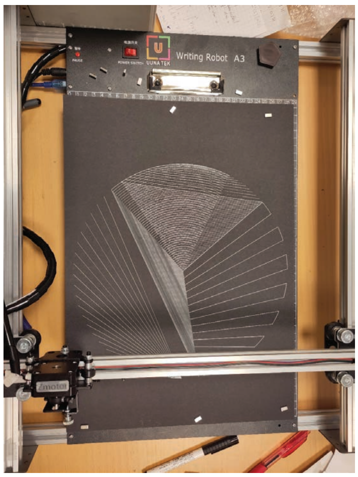
FIGURE 1. An early pen plotter (Calcomp 565). (Source: Amy Goodchild, https://www.amygoodchild.com/blog/computer-art-50s-and-60s.)

```
function moon(x1, y1, x2, y2) {
  d = x2 - x1
  cx = rm + d /2
  cy = tm + (y2 - y1)/ 2
  amin = random(42,96)
  astart = amin
  amax = random(121, 177)
  one(cx, cy, d, amin, amax)
  amin = amax
  amax = amax+random(42, 126)
  onet(cx, cy, d, amin, amax)
  amin = amax
  amax = amin + random(126, 168)
  two(cx, cy, d, amin, amax)
  amin = amin + random (3,11)
  three(cx, cy, d, amin, amax)
  amin = amax
  amax = astart+random(42,63)
  four(cx, cy, d, amin, amax)
}
```
(a)



(b)

```
G21 ; Set Units to Millimeters
G17 ; Set Plane Selection to XY
G90 ; Set Absolute Positioning
F2500 ; Set Speed to 2500 mm/min
G0Z0 ; pen up
G00 X198.6360 Y80.8276
G0Z6 ; pen down on paper
G01 X37.3865 Y126.8931
G0Z0 ; pen up
G00 X197.9154 Y82.6867
G0Z6 ; pen down on paper
G01 X38.9805 Y128.0909
G0Z0 ; pen up
G00 X197.1948 Y84.5458
```
(c)



(d)

FIGURE 2. Programming for digital and physical art. (a) Programming a generative, digital artwork (with p5.js). (b) An image rendering one instance of the generative art program in (a). (c) Programming the drawing machine to physicalize the artwork (G-code) in (a). (d) Output on the drawing machine.

artists. Figure 2(d) shows an example of a modern pen plotter. Modern electronics drive fine servomotors, which support the creativity of makers who can draw a wide range of art genres. Such electronics power more exotic drawing machines. For example, sand machines move a ball on a sand surface, drawing patterns that can be forever erased and regenerated.[6] A spectacular recent example of interaction among programming, machines, and drawing can be seen in the art practice of Sougwen Chung[7]: she programs robots to perform collaborative live painting, where she and the robot collaborate to create an artwork together.

Why do algorithmic artists love drawing machines? First, it frees artists from the tedious task of crafting a physical piece, letting them fully concentrate on creativity.[8] Second, programmable machines equipped with state-of-the-art electronics and motors can draw with a precision that cannot be reached by human operators. Third, drawing machines can work for dozens and even hundreds of hours, tirelessly, to create extremely sophisticated pieces. Finally, there is pure enjoyment in the act of watching a machine perform aesthetically appealing tasks with high precision and diligence (see https://www.youtube.com/watch?v=VpApU07VTV0).

Drawing machines, which physicalize code execution, also provide an excellent opportunity to engage with beginner programmers.[9] Students share the same enjoyment as artists when they see a machine executing their program. The presence of a computer bug can be felt physically, with a wrong drawing. The notion of a loop can be understood visually, by looking at the drawing machine arms making the same pattern again and again.

The notion of a loop can be understood visually, by looking at the drawing machine arms making the same pattern again and again.

FIGURE 3. Early programming art from 1969, rendered on a line printer. (Source: Department of Computer Science at the University of Regina, https://ur50.cs.uregina.ca/?p=176.)

PROGRAMMING ABSTRACTIONS

Drawing with machines usually relies on multiple levels of programming abstractions.[10] Artists use high-level languages to design the artwork, its geometry, and its overall design and texture. A low-level language, on the side of the stack, is needed to control the drawing machine. Depending on the choice of abstractions, the refinement from one level to the other will go through different intermediate representations.

Algorithmic artists can rely on a wide variety of software libraries for making art. Processing in Java, openFrameworks in C++, p5 in Javascript, and nannou in Rust all provide APIs to support artists with the programming of generative artworks. They provide primitive functions to draw basic geometric shapes, such as circles and quadrilaterals, as well as more complex objects such as polygons and Bezier curves. Their API includes the essential mathematical functions to compose and distribute objects on the canvas, such as sine, distance, or vectors. At the core of generative art, randomness plays a key role, and these libraries provide various ways of generating and controlling randomness, from pure random generators to more elaborate noise functions such as simplex or Perlin noise.

On the other hand, the drawing machine does not speak Java or any high-level programming language. Drawing machines require low-level code. This code is about precise control of the pen's position, about tuning the speed for the motors, about fine-grained synchronization among the motors. One of the most popular languages at this level is called G-code.[11] This programming language was originally designed for computer numerical control (CNC), and its application has expanded to 3D printing, additive manufacturing,[12] and pen plotting. It defines an instruction set that can be interpreted by a machine controller to orchestrate the velocity and movements of the motors. For example, let us consider again the image in Figure 2(b), which is generated by the program in Figure 2(a). Figure 2(c) shows the low-level code that steers the drawing machine to create the artwork.

Essentially, there is a need to translate high-level languages used by the artists into the low-level language used to control the drawing machine. For example, one needs to transform the p5 program of Figure 2(a) into the G-code of Figure 2(c). One standard method is to use scalable vector graphics (SVG), an image format in XML for defining 2D graphics, as an intermediate language between the high- and low-level representations of the drawing. In our context, this means a generative art program in a high-level language such as p5 is first translated into an SVG file, and then the SVG is translated into G-code, which is sent to the pen plotter. Libraries support artists for that translation step, such as the state-of-the-art juicy-G-code.[13] Those libraries can also optimize the G-code before plotting, in order to remove redundancies, simplify the paths to be drawn, optimize the curves, etc.

Notably, drawing the execution on a physical medium moves away from pixel-perfect art pieces that we see on screen. The amount and direction of the ink deposited on the paper varies depending on the speed, direction, and type of the drawing pen. This means that the compiler and optimizers used to transform the high-level language into low-level G-code have an influence on the final rendering and texture of the art piece. More than this, various glitches result from the machines' motors, from the texture of the surface on which the machine draws, or from the viscosity of the ink used to draw. Certain artists love this part, considering these hard to predict variations as another source of valuable randomness for the piece. Here the physicalization procedure becomes an integral part of the art itself. When fully embraced, this is even called "glitch art."

OTHER USAGES

The frontiers of drawing machines lie in the very meaning of "drawing." In the world of machinery, CNC machines can automate drilling, milling or carving. Originally made to create mechanical pieces for further assembly, artists have also incorporated CNC machines as part of their practice for the physicalization of algorithmic art works, yielding generative sculptures or engravings. From a programming perspective, controlling the laser or the spline in a 2D space is strictly equivalent to controlling the pen. As a matter of fact, the G-code language to control the machine is the same, and drawing and control libraries are reused across domains, with extensions from art to manufacturing and vice versa.

Light installations are made of multiple programmable light fixtures and lasers. By lighting them up in sequence, moving them in space, and changing their color, light artists "draw" in space and create visual emotions. The programming tools for light and pen plotter control are different, yet the conceptual alphabet of different kinds of repetition and randomization is profoundly similar.

Also based on light, drone art consists of drawing shapes in the sky with thousands of programmable drones. For example, one can drone-paint an eerily moving picture in the sky of Central Park (https://www.youtube.com/watch?v=Dpgy02OB4zE). The programming of drone art is heavyweight, requiring the programming of multiple trajectories in 3D, while handling caveats related to wind and synchronization, which typically do not occur with a pen plotter.

The space of programming to physicalize algorithmic art is open ended. As a final example, imagine 504 analog small clocks with two programmable hands each. This means that one can control 1,008 angle values at each point in time. With very small time steps and medical precision motors, this allows for programming mesmerizing evolving patterns. This is the core expertise and profitable business of the Stockholm design studio Humans since 1982. The core of this high-end design product is definitely a program, while the drawing machine is a specially crafted for a unique physical experience.

In this column, we tied the worlds of art, programming, and machines together. We shed light onto the software technology (programming languages, libraries, and runtime) that fuels algorithmic art. We showed that the physicalization of algorithmic art creates unique challenges and rare opportunities for unprecedented artistic emotions. The growing interest of art institutions and private collectors for algorithmic art is the invaluable energy to support further development of this technology. Undoubtedly, computer programs and drawing machines will continue to augment the artists' visions and abilities. C

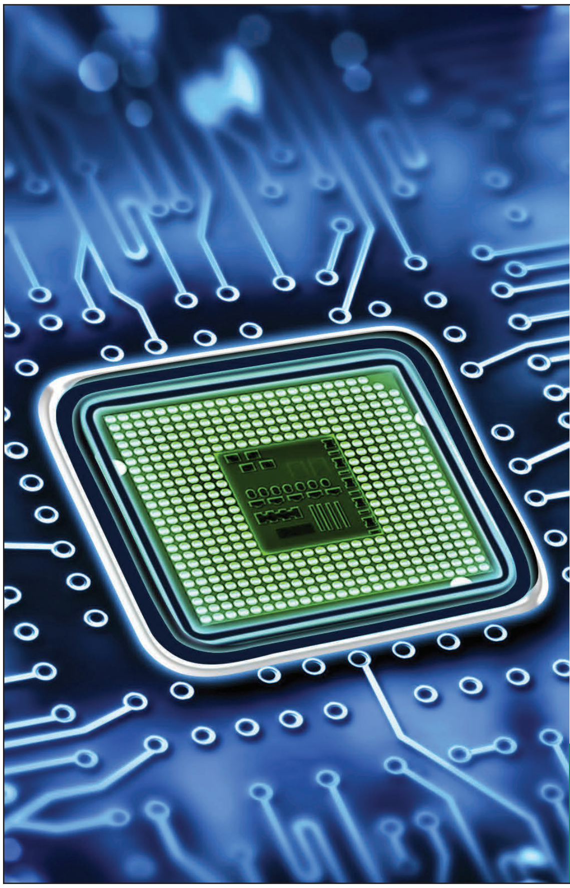
REFERENCES

1. V. Molnar, "Toward aesthetic guidelines for paintings with the aid of a computer," *Leonardo*, vol. 8, no. 3, pp. 185–189, 1975, doi: 10.2307/1573236.
2. P. E. Ceruzzi, "Jacquard's web: How a hand-loom led to the birth of the information age," *Technol. Culture*, vol. 47, no. 1, pp. 197–198, 2006, doi: 10.1353/tech.2006.0061.
3. E. O'Hanrahan, "The contribution of Desmond Paul Henry (1921–2004) to twentieth-century computer art," *Leonardo*, vol. 51, no. 2, pp. 156–162, 2018, doi: 10.1162/LEON_a_01326. [Online]. Available: https://direct.mit.edu/leon/article-pdf/51/2/156/1578147/leon_a_01326.pdf
4. H. W. Franke, *Computer Graphics—Computer Art*. Berlin, Germany: Springer Science & Business Media, 2012. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-70259-4
5. F. Dietrich, "Visual intelligence: The first decade of computer art (1965–1975)," *Leonardo*, vol. 19, no. 2, pp. 159–169, 1986, doi: 10.2307/1578284. [Online]. Available: https://muse.jhu.edu/article/600927.
6. S. Türk and K. Müller, "Kinetic art table: Polar sand plotter," BSc thesis, KTH Royal Inst. Technol., Stockholm, Sweden, 2021. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1559481/FULLTEXT01.pdf
7. C. M. Fang et al., "Machines as collaborators for art and rituals: An interview with Sougwen Chung," MIT Libraries, Cambridge, MA, USA, 2023. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/151089/3596928.pdf
8. R. Verostko, "Epigenetic painting: Software as genotype," *Leonardo*, vol. 23, no. 1, pp. 17–23, 1990, doi: 10.2307/1578459. [Online]. Available: https://muse.jhu.edu/article/601786/pdf

9. S. He and J. Jones, "Collaborative creative coding through drawing robots," in *Proc. 16th Int. Conf. Tangible, Embedded, Embodied Interaction*, 2022, pp. 1–4, doi: 10.1145/3490149.3503667.
10. N. N. M. Peek, "Making machines that make: Object-oriented hardware meets object-oriented software," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 2016. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/107578/974648092-MIT.pdf
11. G. Bowers, "Machine tool control via a minicomputer," Oak Ridge, TN, USA, Tech. Rep. Y-1870, 1973.
12. A. Gleadall, "FullControl GCode designer: Open-source software for unconstrained design in additive manufacturing," *Additive Manuf.*, vol. 46, Oct. 2021, Art. no. 102109, doi: 10.1016/j.addma.2021.102109. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214860421002748
13. "Juicy-GCode." GitHub. Accessed: Feb. 12, 2024. [Online]. Available: https://github.com/domoszlai/juicy-gcode

BENOIT BAUDRY is a professor of software engineering at Université de Montréal, Montréal, QC H3T 1J4, Canada, and KTH Royal Institute of Technology, 10044 Stockholm, Sweden. Contact him at benoit.baudry@umontreal.ca.

MARTIN MONPERRUS is a professor of software technology at KTH Royal Institute of Technology, 100 44 Stockholm, Sweden. Contact him at monperrus@kth.se.



IEEE TRANSACTIONS ON

COMPUTERS

Call for Papers:
IEEE Transactions on Computers

Publish your work in the IEEE Computer Society's flagship journal, *IEEE Transactions on Computers*. The journal seeks papers on everything from computer architecture and software systems to machine learning and quantum computing.

Learn about calls for papers and submission details at **www.computer.org/tc.**

IEEE COMPUTER SOCIETY IEEE

Digital Object Identifier 10.1109/MC.2024.3408525