Light Pattern: Writing Code with Photographs

Daniel Temkin

Daniel Temkin
Artist
4301 22nd St., Studio #238
LIC, NY 11101
daniel@danieltemkin.com

ABSTRACT

This paper explores the author's Light Pattern project, a programming language where code is written with photographs rather than text. Light Pattern explores programming languages as the most direct conduit between human thinking and machine logic. It emphasizes the nuance, tone and personal style inherent in all code. It also creates an algorithmic photography structured by the programs one writes, but not ultimately computer-generated. The paper looks at connections to both hobbyist/hacker culture (specifically esolangs) and to art-historical impulses and movements such as Fluxus and Oulipo.

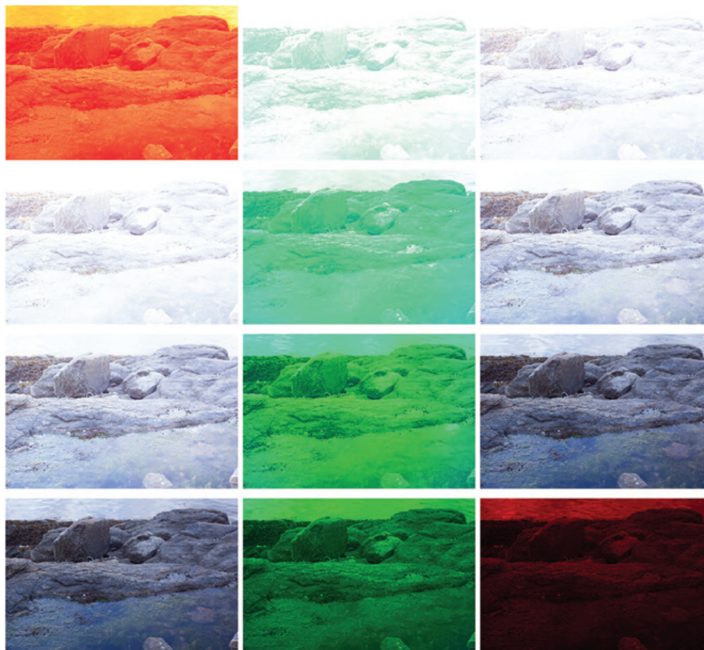

Figure 1. Photographs for a "Hello, World" program (2014), size variable. Resulting program can be seen here: https://vimeo.com/102076781. © 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

Light Pattern is a programming language in which code is written in photographs. To see the source code of Light Pattern, one doesn't open a text file with lines of code of the "goto 10" variety. Instead, one views a directory full of JPEGs to be read in alphabetical order according to their file names. The images could be vacation photos or yearbook images. However, they hold a secondary form of information: the change in exposure and dominant color from one image to the next determines commands in the Light Pattern language. Collectively, they might be a "Hello, World!" program, or perhaps (with several hundred thousand images) a working web server.

All code, in any language, has affect—content apart from what's necessary to make the machine do what we want. Light Pattern brings this to the forefront: here nuance, tone and personal style

doi:10.1162/LEON_a_01091

© 2015 Daniel Temkin

are the most visible aspects of the code, in opposition to the mechanical, objective voice that coders often aspire to adopt. In Light Pattern, these ordinarily secondary meanings become the most visible. Content for the compiler is encoded in patterns in the appearance of the series of images, but the most salient messages of the images—from the images' subjects to their composition—are irrelevant to the machine and the language.

Light Pattern plays off conceptual photographic works that underscore the indifference of the camera to human reading of the photograph, as in John Hilliard's "Camera Recording Its Own Condition (7 Apertures, 10 Speeds, 2 Mirrors)" (1971). In this work, Hilliard's camera photographs its own reflection repeatedly, in a grid of exposures ranging from the vastly overexposed to the completely black. The camera remains immobile from one image to the next, yet the readability of the images ranges from the well-exposed in the center of the spectrum, to the contentless at the ends.

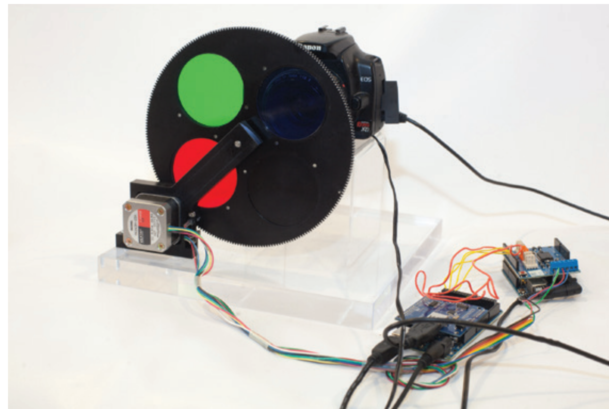

Figure 2. The Light Pattern machine (2014). 12" x 10" base. Two Arduinos, a filter wheel and Canon EOS camera. Code to control the machine can be found here: <https://github.com/rottytooth/LightPattern_Arduino>. © 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

In Light Pattern (LP), the indifference of the camera is equated with the indifference of the computer. Light Pattern's Recursive Program references this directly. We replaced the camera of Hilliard's piece with a slightly more complex system (Figure 2). A Canon EOS camera sits behind a motorized filter wheel scavenged from a laser system; each is controlled by an Arduino. The filter wheel is fitted with colored lenses (red, green and blue), as these—corresponding to the components of pixel color—are the colors Light Pattern looks for in an image. The two Arduinos control the filter wheel and camera, setting this camera-machine to photograph itself in the mirror. The images create a recursive fork bomb—a program that launches a copy of itself, and then the copy launches another copy, and so on until the machine crashes. By recording its own image, the machine creates a program that invokes itself, creating two closed loops.

This is analogous to writing a C program with ridiculous variable names and long, off-topic comments, to make the affect of the code its primary message, over clarity of the code's behavior. This is done sometimes in so-called Obfuscated Programs, which create seeming nonsense in code, making the programming language visible in a way it ordinarily isn't. Instead of seeing "through" the code to take in the algorithm it represents—what we expect in well-written code—the code becomes all we can see [1].

Esolangs and Program Recipes

Light Pattern is an esolang or "esoteric programming language," a form invented by hobbyists and programmers that uses programming languages for unconventional purposes. The language Whenever, for instance, has lines of commands that might be run in any order by the compiler; the programmer writes code without knowing in what order it will be executed. Another language, Malbolge, is so difficult to write code in that the first program for it appeared a full three years after it was introduced (the program was the classic "Hello, World"). It, like all

Malbolge programs, was written by another program: only by treating the language as an encryption problem could it be "solved" to write a useful program. Ben Olmstead, creator of the Malbolge, describes his work and those of other esolangers as "pushing the boundaries of programming, but not in useful directions" [2].

My blog, esoteric.codes (supported by Creative Capitol and the Warhol Foundation), studies the history of these languages and their engagement with programming languages as art. As part of this research, I looked at what we might think of as "logic-based esolangs," those like Whenever and Malbolge, which pose questions or challenges about programming. The aesthetic of esolangs tends to favor this approach. Ais523, an active esolanger who is currently responsible for maintaining C-INTERCAL, the most widely used implementation of the first esolang (created in 1972), puts it this way: "Don't get hung up on syntax; what a language looks like is trivial to change simply with a find-and-replace, and it's more important to have an interesting computational model" [3].

However, vocabulary-oriented esolangs can also offer a rich approach. This has been shown by languages like Piet, where code is written in images, often resembling the works of Mondrian (for whom the language was named). David Morgan-Mar, the creator of Piet, spoke of Whenever, which is completely logic-oriented, as the work of which he's most proud, despite the popularity of Piet. In Light Pattern, I wanted to fully embrace the vocabulary-oriented approach and take it as far as possible, using it to create a rhizomatic exploration of photography and code [4].

To do this, Light Pattern had to both allow for endless expansion and reinterpretation and create programs where the code and its expression were related. My solution was to create "program recipes": programs described by their Light Pattern constraints and an idea to be explored in the photos, to then be implemented repeatedly in actual photographs, each one its own photo series. To write a Light Pattern program requires shooting photos according to the constraints of the language. It is designed around the Oulipean goal, as described by Queneau: to become the "rats who build the labyrinth from which they will try to escape" [5]. A series of shutter speeds, apertures and dominant colors can be determined ahead of time for a photographer to then carry

Figure 3. Still from the rot13("Hello, World") video (2014), 1080p. The video can be seen here: <https://vimeo.com/102190669>.
© 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

out. Each recipe creates its own set of constraints for a program, determining a framework for the relationship between image and code, a language inside of the language.

Hello, World

To enact this program recipe idea, I began shooting the Hello, World program as a daily shooting practice. Each day, between 11am and 1pm—generally considered the worst time of day to photograph due to the high contrast—I shot the 51 photos necessary to write the "Hello, World!" for my particular camera and lens. I shot at this time of day because the high contrast ensured that even the brightest and darkest photos would each get enough color content for the compiler to read (shades of black are okay—even one slightly greener pixel in an image would work to make the image read as "green," but if it were truly all white or all black, the compiler would not be able to determine color and so would skip the image). I held color filters over the lens with each image, to be able to swap them quickly. I took the resulting programs and made each a video, with exposures cascading over the previous ones in sequence. Each transition had a floating title with a three-digit ternary number corresponding to the change in exposure. These numbers pile up on the right side of the screen, eventually constructing commands: building a string, adding "H," adding "e," etc., until the exposures get too dark. At that point, the string ends, the code for "ignore the next image" is fired, and a much brighter image follows. Then, back to the work of building the "Hello, World!" string to be printed.

The result was a series of odd, yet expressly banal image sequences. An unreadable, monochrome red, seemingly underwater scene is revealed as a shot of rocks and seaweed with a water backdrop. Blue and green stripes show a narrative of garbage swirling in a puddle in a suburban dump. A radio tower site is populated by people with backpacks who seem to just move back and forth in a sort of fugue. Kill Screen Daily, a video game blog, described them as expressing "a strange tension between the art of photography and the rigid demands of programming" [6]. The high contrast tends to make for unflattering imagery. Meanwhile, because it is difficult to shoot

Figure 4. Composite image from the Three Lamp Events program (2014), still from 1080p video. The final video can be seen here: <https://vimeo.com/123002470>. © 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

all the images in the right order, I often make mistakes. The resulting videos for these contain mistakes ranging from a wrong letter ("Hfllo, World!") to a pile of error codes when a syntax error is invoked. These failed programs are presented alongside the working programs; each is shaped in an attempt to communicate with the machine, but only the fact that they are shaped by this attempt is important; their success is secondary.

Esolangs and Fluxus

Esolangs are participatory forms: anyone can program in them. They are also natively open source; to make a language available for others to use means exposing the rules for the language. A language is just a set of rules, nothing more. I see this dematerialized quality and participatory style similar to that of Fluxus, especially in the event scores. In order to illustrate this connection, I made a Light Pattern program based on George Brecht's *Three Lamp Events*. Brecht was interested in annihilating the line between art and life. His event score *Exit* (with a single instruction: "exit") was spoken of by George Maciunas as the ultimate Fluxus work: one that would be performed constantly, without the performer necessarily even knowing that a Fluxus performance was occurring.

Light Pattern, which uses photos—even incidental photos (if they are arranged correctly)—to function as code, borrows from this Fluxus playbook. The Light Pattern program "Three Lamp Events" recreates Brecht's score of the same name. Brecht's piece orders us to turn a lamp on and off. This is the entire score of the 1961 version:

> on. off.
> lamp
> off. on. [7]

In the Light Pattern "Three Lamp Events," three different lamps are needed; one with a red bulb, one with green, and one with blue, in order to get the correct dominant color for each image. As the exposures of the lamps change from one photo to the next, each lamp appears to grow relatively brighter or dimmer (Figure 4). Together, they print each letter of the score, recreating Brecht's score using the lamps themselves to "write" it.

Light Pattern's Syntax

A programming language has two syntaxes. The concrete syntax is the surface of the language, the actual commands as they might be typed (or photographed, in this case) by a programmer. Below that is the abstract syntax, the tree of commands and expressions stripped of their appearance. Looking at the abstract syntax tree, we can get a sense of the logic of the language: what kind of language Light Pattern is, ignoring its unusual method of input. We can think of the vocabulary-oriented approach discussed earlier as focused more on the concrete level, with the logic-oriented languages often focusing more on the abstract level.

To understand Light Pattern's approach to syntax, we can look at the language Piet in contrast. Piet translates codels—blocks of color as small as one pixel—into code [8]. While Light Pattern uses photos instead of pixels, like Piet it uses deltas (changes) rather than assigning static meanings to individual photos. In the case of Piet, changes in hue and in brightness are like phonemes, units of language that in combination create commands. In Light Pattern, aperture, shutter speed and color take on this role—these were selected because they are easy for the programmer to influence, while having a clear influence on the look of the image. A second dialect of Light Pattern extends the language to photos without embedded EXIF data, using brightness and saturation in place of the first two elements. While there are degrees of change in

Figure 5. Still from a "Hello, World" video (2014), 1080p. The final video can be seen here: <https://vimeo.com/102190632>. © 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

brightness in Piet, Light Pattern doesn't care how much the aperture changes, only in which direction; this is also done to make Light Pattern easier to use, as producing a series of photos can be time-intensive. The aperture change is expressed as a single ternary (base three) digit: 0 for smaller (darker), 1 for no change, 2 for larger (brighter). Color and shutter follow a similar format, creating a three-digit base-three number for each transition between photographs.

Piet was influenced by Chris Pressey's Befunge language, the 2D language that used characters, rather than pixels, as commands, but which similarly allows program flow to go back and forth in many directions across a plane. Befunge (like Piet) used a stack-based model, influenced by FALSE (one of the first esolangs, passed around on Amiga floppy disks in the early 90s), which was influenced by FORTH, an oddball language enormously influential to esoteric programming. Stack-based programming provides a strategy for keeping each command very small; we can access data from stack location, meaning we never have to "name" variables. It is elegant in its simplicity and allows for languages with compact vocabularies [9].

Despite the appeal of that approach, Light Pattern has more of a utilitarian outlook. It's a traditional procedural language modeled on Java and C#; corporate and bland, familiar to nearly anyone who has been exposed to programming. In part, this was to keep the focus on building the series of photos, rather than figuring out stack operations. It squarely situates Light Pattern as vocabulary-oriented. Second, fewer photos are necessary to build a program. In Piet, if we want to write the letter "H" to the screen, we need to add the number 1 to the stack, double, it, multiply it by itself, performing a repetitive series of simple math operations to get to 72, the Unicode number for "H." In Light Pattern, we translate 72 into ternary (base 3), which can be expressed in two or three photos. Light Pattern challenges the aesthetic often sought in esolangs—of building a logically elegant language—in order to keep the focus on the front end: the relationship with the photographs used to create the works. Its similarity to C# (in which it is written) also makes it easier to convert to similar, C-based languages. Its current compiler can generate C# or JavaScript code from Light Pattern and could be adapted for Java or other similar languages easily.

Future Plans

This experiment in relating code and photography is only made concrete through programs written in the language and through this new approach of using recipes to shape bodies of work. The results do not need to be presented on a computer; the programs I have presented in this paper are often presented in video format, or as a series of prints (Figures 5 and 6). Chris Pressey once described esolangs as "made up of concepts, and these concepts would exist even if our computing equipment wasn't electronic, or wasn't digital, or if we didn't have computing equipment at all. It's just that having computing equipment makes it a lot easier to design and experience these programming languages." Light Pattern programs, likewise, can be presented in non-digital contexts as easily: the photos used in these programs carry that information in their very images [10].

Figure 6. Photos for the first test program in Light Pattern (2011), sizes variable. © 2014 Daniel Temkin. Released under a Creative Commons Attribution 3.0 Unported License.

Light Pattern is the first programming language to actively bridge the Oulipean and Fluxus impulses with esolangs, a connection which was always latent, but never engaged directly. It is also the first to be exhibited in a non-digital context: away from the computer, in video or still photo formats, emphasizing the bodies of work that present the esolang as a conceptual framework. My hope is that this avenue of discovery will continue to lead to a richer approach in using programming languages as an art medium.

References

1. Nick Montfort, "Obfuscated Code," in *Software Studies*, ed. Matthew Fuller, 196.
2. Ben Olmstead, "Interview with Ben Olmstead," esoteric.codes, published 12/16/2014, <http://esoteric.codes/post/101675489813/interview-with-ben-olmstead>.
3. ais523, "Interview with ais523," esoteric.codes, published 2/28/2011. <http://esoteric.codes/post/84454956003/interview-with-ais523>.
4. David Morgan-Mar, "Interiew with David Morgan-Mar," esoteric.codes, published 02/10/2015, <http://esoteric.codes/post/110647356808/interview-with-david-morgan-mar>.
5. Jean Lescure, "A Brief History of the Oulipo" in *Oulipo: A Primer of Potential Literature*, ed. and trans. Warren F. Motte Jr, p. 37.
6. Zach Budgor, "Arbitrary Subjects Turn into Kaleidoscopic Imagery with Light Pattern," *Kill Screen Daily*, published 08/06/2014, <http://killscreendaily.com/articles/kaleidoscopic-coding-programming-language-light-pattern/>.
7. George Brecht, "Three Lamp Events," *The Fluxus Performance Workbook*, ed. Ken Friedman et al., p. 23. Performance Research e-publication.
8. David Morgan-Mar, "Piet," last updated 5/28/2014, retrieved 4/3/2015, <www.dangermouse.net/esoteric/piet.html>.
9. Chris Pressey, "Interview with Chris Pressey," to be published on esoteric.codes.
10. Chris Pressey, "The Aesthetics of Esolangs," published 6/17/2013, retrieved 1/20/2015, <http://catseye.tc/node/The_Aesthetics_of_Esolangs>.