Programming in Nature

Resumo

This paper describes current work-in-progress in a practice I call "Programming in Nature" which sits somewhere between art and computer science. It draws on elements of landscape intervention, photography and "technoshamanic" ritual, computer vision, experimental programming languages (in particular esoteric programming languages (esolangs)), physical computing and user-interface design.

The aim of Programming in Nature is to be able to do computer programming away from the traditional office environment, keyboard and screen, and to perform it out-of-doors in a natural landscape of "cerrado", woodland and riverside. I invent programming "languages" not made of words but of assemblages of natural objects which can be interpreted by a computer and compiled into executable software.

This paper discusses the the motivations for Programming in Nature, its influences and the current state of development of various components.

Why are we unhappy?

I would like to start this with a highly informal, speculative bit of free-form conjecture. The historical and anthropological, even spiritual, merits of this speculation can be argued, but the rest of work is best understood in light of it.

Humans are a language-using animal. And ever since we evolved our linguistic capacities (which include symbolic representations of the world, and symbol-based action within it) we have been keen to substitute language for the other kinds of effort we make to accomplish our daily tasks. The shaman uses symbols and ritual practices to "negotiate" with the animal who is to be

hunted, because talking to it is, in some sense, "easier" than running around chasing it with a spear.

What we call "magical" thinking, our religions, our "shamanic" practices etc. are all examples of this very human aspiration : that the world can be addressed through symbols and entreaties and communication, rather than manhandled with a tiring brute force.

Over recorded human history we have actually succeeded in extending the reach of language; such that our modern lives are largely lived in a world of symbols. Many of us read and write for a living. As individuals, we produce a fraction of the food or goods we consume. Most of what we acquire is through manipulating symbols and navigating flows of extraordinary abstractions that would astound our ancestors.

In this sense, the contemporary world is made of magic.

So why are we still unhappy?

We are unhappy because the transformation of the world into an infrastructure of ideas comes at a high cost to the individual : bureaucracy, hierarchy, markets, finance, discipline, control, impersonalization, atomization, alienation and all the usual ills attributed to industrial capitalism. In contrast, in our idyllic reconstructions of spiritual worlds, magic is the result of self-directed research, of powerful inner essence, of individualistic expression, or of small-scale communal ritual that harmoniously blends work, leisure, social bonding and the rhythm of life. In other words, our conception of magic is a reflection of our ideal of how we should work and produce.

We are unhappy because, as Bret Victor points out, engagement with the symbolic order has assaulted our bodies and senses, placing them in drab, unhealthy offices, sitting for hours hunched over "tiny rectangles"[1] of paper or screen, restricted to micro-movements of fingers wagging a pen or fluttering on a keyboard. We have lost the majority of modalities that our bodies and senses provide, and which we perhaps expect in life. We have lost the connection with the landscapes and environments in which we evolved. The forests and lakes and rivers and mountains, and replaced them with concrete cages. We are obese and unhealthy.

Furthermore, our society firmly demarcates work and production from leisure and consumption. In work, we live a genuinely "magical" experience, where objects can be conjured into existence and flown around the world with a

[1] @victor_humane

few keystrokes (or symbolic invocations of the global supply-chain). But *aesthetically*, the magic that governs the modern economic system is hostile and inhuman.

At the same time, our leisure and culture reveals the aesthetic of magic and the spiritual that we actually crave. Churches to invisible deities are packed. The streets are filled with Pokemon Go players, hunting fantastical beasts that can only be seen with the aid of enchanted lenses. Super-hero movies dominate Hollywood; fantasy literature is our common culture. Even the most technologically knowledgeable artists can seem obsessed with esotericism and pseudo-science and railing against enlightenment rationality.

Art tells us what we want. But as long as this desire is relegated the sphere of consumption, as long as it is our "reward" for participation and collaboration in the contemporary mode of production, then it can have little chance to rescue us from our predicament. It has no power to effect change in our economy and, therefore, our society.

Instead, we must infect the realm of production with the magical aesthetic.

We need to get out more.

The above is necessarily sketchy and superficial, but it forms the tangle of intuitions and inspirations behind this project.

Programming in Nature is a practice, or a series of experiments, which fall somewhere between fringe computer science, and a range of artistic and political concerns / practices which I associate with the movement called "technoshamanism"[Fabi].

It is highly influenced by the thinking of Bret Victor on "humane" computing [2] who invites us to start speculating on and designing for a way of using and interacting with computers which features all modalities of the body (visual, aural, tactile, kinaesthetic and spatial) and thought (symbolic, iconic, enactive).

It should also be understood within the tradition of John Ruskin and William Morris and the Arts and Crafts Movement, who launched an attack on the industrial revolution from a Romantic reconstruction of an idealized (and perhaps imaginary) mediaeval model of work and craftsmanship. In many ways, Morris can be seen as founder of this project. Even though

[2] @victor_humane

the particular aesthetic and the kinds of production we deal with are very different from his.[3]

We choose to ask a particular question : "what if we could do programming *outside*?" That is, outside buildings, away from the traditional screens, keyboards, offices. Away from the aesthetic style of contemporary bureaucracy. And within the aesthetic style that we seem to be attracted to? In the forest; by the pool in the river below a waterfall; working with the natural materials we find there?

To be clear about the goal. *"Programming"* implies creating real computer programs, which can be compiled and executed on a real computer. And today, this implies doing almost any "work" within the contemporary economy, where software mediates most productive activities. *"in nature"* is intended to signal a minimization of the "props" or style of the contemporary economy. It conjures images of druids in sacred groves, fauns, dryads, magical springs and other appealing images. (Though one can also imagine ceremonies in dark caves and witches cackling over cauldrons on the stormy heath.) What if we could live and work like that?

Of course, this is all a distant and ambitious goal. But in pursuing it I believe we will discover much of interest along the way. The rest of this paper describes the some early experiments in this practice.

Programming with Camerast

Our initial approach can be described as "Programming with Cameras". Smart-phones that incorporate digital cameras are now ubiquitous. They are cheap, small and robust enough that people habitually carry them into the natural landscape. But we want to avoid using the screen for input or much interactivity. We do not want to take our "tiny rectangles" into the woods. We ought to be able to make programs out of the physical items we find there, in a practice that engages our whole body. The camera is used simply to capture these physical assemblages of objects so that we can feed a sequence of photographs to the computer to be analysed with computer vision algorithms, and then interpreted and parsed into code.

Programming requires a programming "language". That is, a set of meaningful symbols with some grammatical constraints that join them together, and

[3] Ironically, hypertext entrepreneur Mark Bernstein once launched a modern equivalent of the Arts and Crafts movement he called "neo-Victorianism", where he celebrated the Victorian workshop aesthetic.

a “semantics” a way that a meaning can be assigned to the symbols. When we consider a language made of computer-interpreted photographs we note one great strength and one great weakness of photographs.

- The weakness is that computer recognition becomes more complex as the number of symbols we wish to distinguish increases. Separating broadly red things from broadly blue things is fairly simple. Interpreting and distinguishing detailed figures is harder.
- The strength is that we have accurate spatial relationships between objects in two dimensions.

Figure 1: This leaf is interpreted as a low-pass filter

We, want, therefore, a language which both tries to minimize the number of distinct symbols that need to be disambiguated. But one which can take advantage of the spatial layout in two dimensions. There are some interesting “esolangs”[4] which are two-dimensional. Perhaps the most famous “artistic” esolang is Piet, a language whose programs are a layout of coloured blocks (alleged to look like the works of Piet Mondrian), but Piet programs require

[4] An “esolang”, or “esoteric language” is a programming language created, often as a joke or artistic work

fairly fine-grained alignments of elements, which may be hard to create out the materials available. And they are hard for humans to write[5]

More promising might be visual / data-flow languages such Pure Data where a number of components are wired together by connections. The very first experiments were done in the context of using PD for electronic music. Images were analysed to extract blobs of colour which could be interpreted as objects within a PD patch. For example, in figure 1 we identify a leaf (and interpret it as a low-pass filter).

Note that all computer vision examples given here are using the free-software OpenCV library.[@opencv]

But a green-blob to low-pass filter mapping is too crude and simplistic. And in these early experiments, the flow from one component to another was simply inferred rather than read from the image. To work with networks we needed to be able to deduce the data-flow topology from the image.

Analysing a Network

The following is a simple example of analysing an image to extract a network structure from it.[6]

A simple flow network of sticks was assembled and photographed. (Fig. 2)

It was then pre-processed with the following algorithm using OpenCV (Fig. 3) :

- Convert to hue-saturation-intensity (HSV) format
- Extract just the saturation plane.
- Normalize the histogram of the image.
- Threshold to get a clearer binary image.

The binary image is then analysed

- Find the contours
- Calculate the enclosing ellipse of each contour

[5] Most Piet examples are actually compiled *from* another language.

[6] I am extremely grateful to *pklab* on the OpenCV Forum for the advice and help with this analysis @pklab__visually__2016

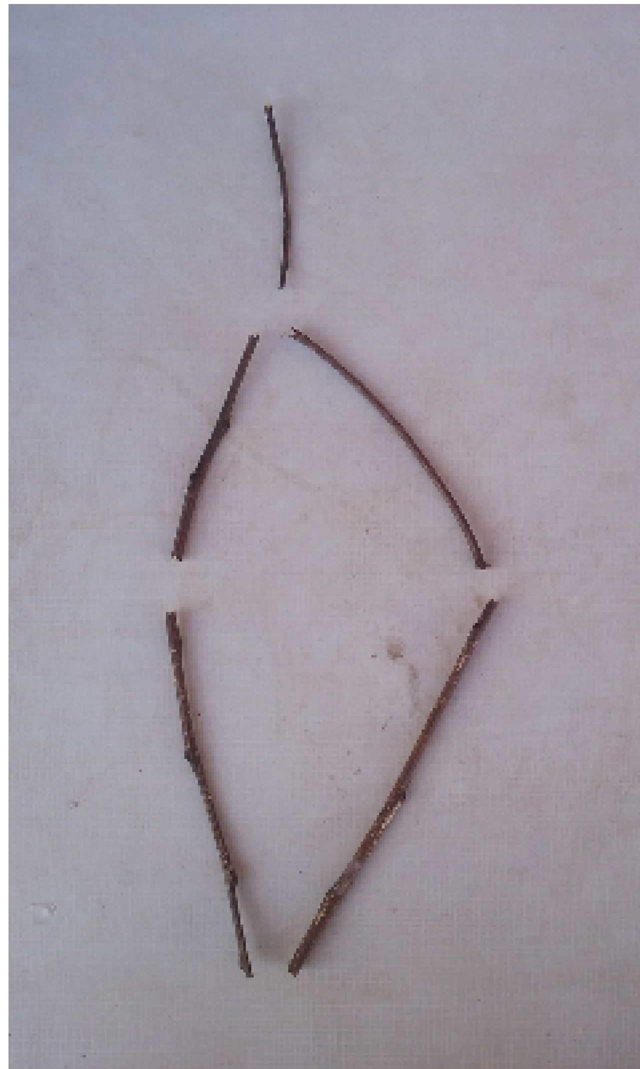

Figure 2: A simple network of sticks

- Filter to preserve only contours that are potential arcs in the network (the criteria is based on size and elongation)
- Calculate a line that passes along the main axis of the ellipse and a pair of end-points for each.

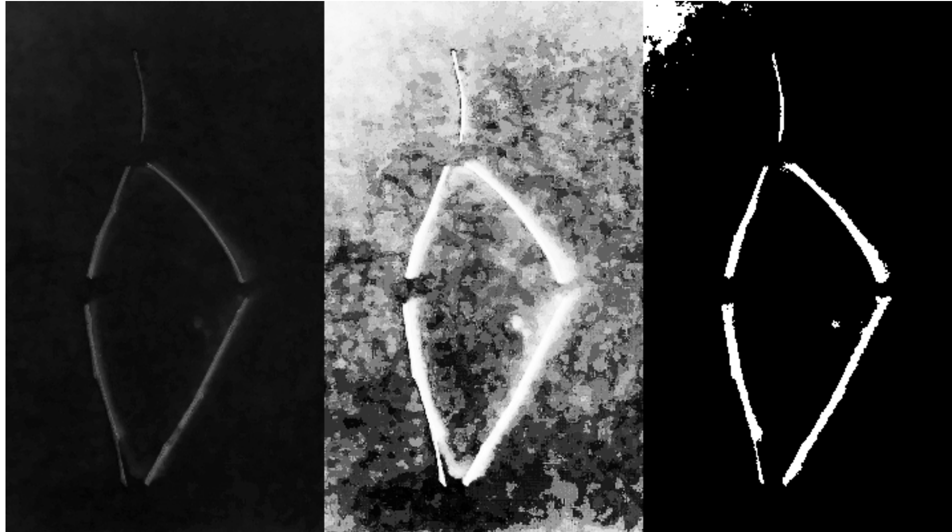

Figure 3: Processing (saturation plane, normalized histogram, threshold)

These lines are now treated as the arcs in a flow network and we calculate the rest of the topology as follows :

- Assume that the two closest end-points in our entire collection must be connected at the same node.
- Multiply the distance between them by a constant to get an estimate of the size of a typical node.[7].
- Check for all points which are within that distance and assume that they are meeting at the same node.
- Extract a list of potential nodes, assuming the centre of a node is the average of all points that are located within it.
- Now calculate which arcs connect which nodes.
- Arcs are considered directional top to bottom. Automatically assume that the top end of an arc is an outflow of the node it is connected to, while the bottom end is an inflow to the node that is connected to.

[7]We estimate two or three times this minimum distance

Figure 4 shows the result of this analysis.

QaSaC

While the analysis described in the previous section gives us a way of extracting networks from images, we still need to describe the nodes which are being connected. PD isn't an ideal language partly because it is specialized for certain kinds of applications and other algorithms are awkward to implement in it. We also remember the other constraint of photographic programming language : that disambiguating objects in an image is hard, and we would like to minimize the number of distinct tokens that need to be distinguished.

QaSaC[8] has been developed as the underlying language and virtual machine for images to be compiled into. QaSaC brings together a PD-like data-flow network with a very simple stack-based concatinative language[@concat], inspired by Forth[@forth] and, particularly, Joy[@joy]. The reason for choosing this type of language is that stack-based programs tend to have fewer named things (such as variables) which we believe reduces the number of distinct tokens that need to be recognised.

Each node in a QaSaC system runs a simple program which consists of an initialization phase and an endless loop (rather like a Processing sketch). The program can pull data from its input queues, push data onto its output queues, and stores and manipulates all other data on a local stack.

A simple example :

```
0 | DUP -> 1 +
```

Note that the vertical bar separates the initialize phase from the looping phase. The program runs as follows :

- `0` is the only instruction in the initialize phase, as a number, it is pushed onto the stack.
- `DUP` is the "duplicate" instruction, it duplicates what is at the top of the stack.
- `->` pops the top value from the stack, and sends the result to one of the output queues.

[8] QaSaC is an acronym for "Queues and Stacks and Combinators", the elements from which the language is made.

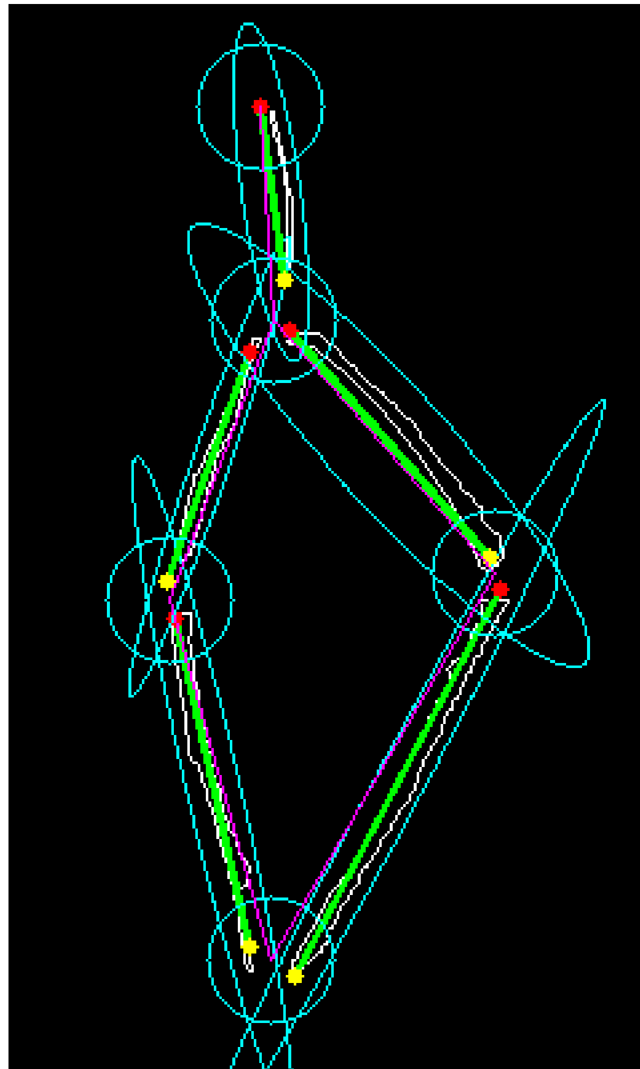

Figure 4: The computer's view of the sticks.

- `1` pushes the number 1 onto the stack
- `+` pops the top two items off the stack (the 0 and the 1), adds them, and pushes the result back onto the stack

This program can be seen to produce the sequence of integers, counting up from zero, on its output queue.

QaSaC uses combinators[@combinators] rather than explicit control structures. These operators take quoted blocks of code from the stack and execute them based on other criteria. For example a filter that only lets even numbers through :

```
X DUP [->] SWAP 2 % 0 = [DROP] SWAP COND
```

The `X` means to pull the next value from the X input queue and push it onto the stack.

`[->]` is a code-block containing a "pop and send to output queue". It's not executed unless the `COND` (conditional combinator) decides it should be.

`SWAP` swaps the top two items on stack. `%` is the "modulus" function. `=` is the test for equality. `DROP` pops the top of the stack and throws the value away. Once again, here, it's in a block that `COND` will decide whether to execute.

Figure 5 shows the whole of a simple QaSaC program containing these nodes in a custom-build editor.

QaSaC, in common with PD and other data-flow languages has a different feel from more "static" languages. For example, a short, simple program can be used to define a reactive user interface[@reactive] if events from mouse or keyboard are fed to it through standard channels. QaSaC is currently implemented in Clojure, and can be run with Quil[@quil], a Clojure wrapper for the Processing API.

Work in Progress and Future Development

Both the network analysis and QaSaC are implemented and can be shown to run[9]. Current work in progress is focused on the next, most important step. Finding a way to parse and interpret images that represent the short QaSaC nodes.

[9] Although some manual tweaking of parameters is required

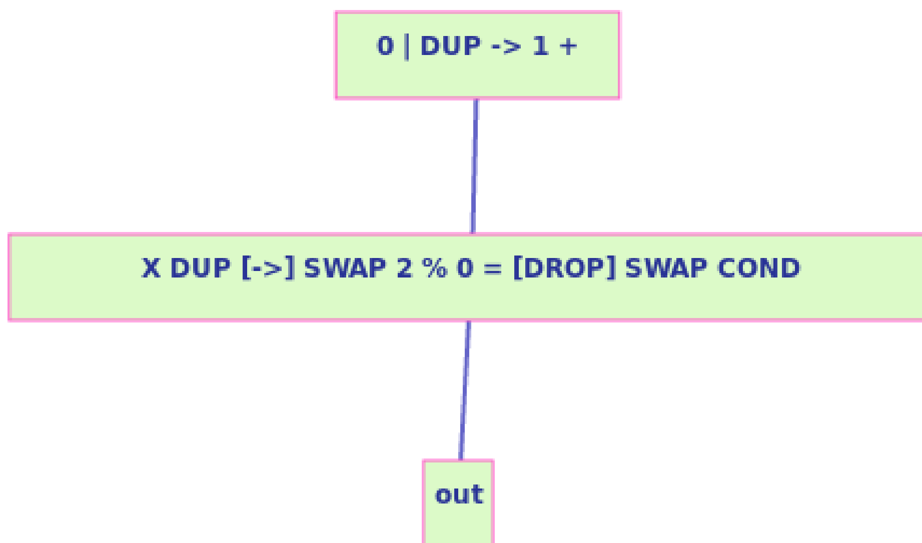

Figure 5: A complete QaSaC program in a simple browser-based editor

One line of research is to try directly using natural materials as in Figure 6. I am experimenting with analysing images like this. But it is hard (and perhaps not plausible) to represent all the QaSaC operators reliably and unambiguously with individual flowers and leaves.

Figure 6: A QaSaC node defined with plants

An alternative is to accept that clear, unambiguous symbols are still necessary and to look for a way to handle this within our aesthetic. We can take inspiration from some uses that are made of alphabets and writing in contemporary esoteric practice. For example, *rune magic*[10] takes runes from pre-christian Germanic peoples and uses them within divination rituals. Runes are fairly clearly distinct symbols designed to be chiselled or scratched into stone or wood. So I am developing an alphabet that combines familiar Latin letters and Arabic numerals with runes and other symbols to represent the specific operations within QaSaC. We will then make physical tokens representing them, and use OpenCV's K Nearest Neighbour classification algorithm, often used for OCR applications, to interpret them.

Our original example of a program that produces a sequence of integers is represented as figure 7.

We need to continue to experiment with both representations and the algorithms to interpret them to have a reliable way of representing these node definitions.

Beyond this, the urgent task is to package this functionality into an Android

[10]http://justwicca.com/magick-runes-3/

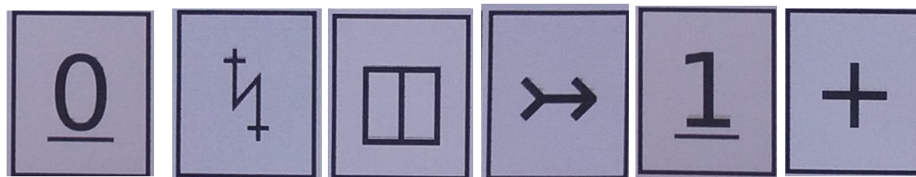

Figure 7: Sequence Generator in a custom alphabet

app. to run on a smart-phone. At present, the computer vision consists of small Python scripts using the OpenCV library. While the QaSaC interpreter / virtual-machine is a separate application written in Clojure. But they need to be fully integrated.

Finally, there is still much to discuss and criticise and explore in this research programme. It is impossible to enter into the many debates in this paper. But I invite more people to join the project and involve themselves in technical development, in the practice of actually creating programs with these tools, and in the many discussions that the project can inspire.

The home-page of the project is at http://pin.alchemyislands.com/ with links to the source-code and details of ongoing work and discussion within the project.

Acknowledgements

I would like to acknowledge several people who have helped and inspired this current work.

Fabi Borges, for promoting the technoshamanic culture, rituals and encounters in which I have participated. These have greatly helped me develop my thinking on technology and a magical style which is given here.

pklab, a contributor to the OpenCV forum who provided extremely helpful and comprehensive advice with some problems I was having with the visual recognition.

Aharon Amir, one of my great artistic inspirations, who has challenged me with his radical thinking and questioning about programming languages.