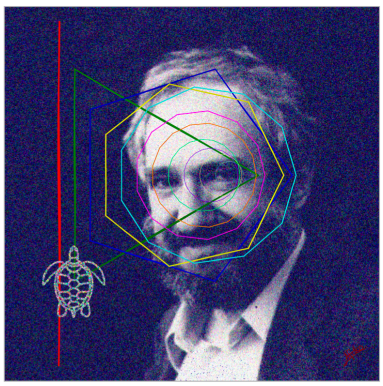
On Making Painting With a Turtle to See More



A computational painting made to celebrate and remember the life of Seymour Papert.

> *"... The modeller works rather like an artist with a painting — she can directly experience and modify the model throughout its construction according to her vision for the model."*
>
> — Construit: Empirical Modelling for Education [1]

By Afriko

Preface

Seymour Papert was born on February 29, 1928, and died on July 31, 2016. He was a visionary, humanist, mathematician, pioneer computer scientist, and one of the first people to demonstrate and promote the use of computers and programming to teach children (and thus adults as well) how to do mathematics, learn to think for themselves, and how to use their imagination in creative and constructive ways.

Painting

Painting, for me, is putting (lasting) colourings onto a solid surface to make seeing experiences for those who look upon the painting. Not particular experiences, nor experiences intended by me, nor even the same experience each time a person looks at the painting: experiences made by the looker from what they see and feel each time they look at the painting. And, it's colourings, not colours, that are put on the painting surface. Colourings are what people look at, colours are what people see: colourings are in the world we look at, colours are part of the conscious experience of seeing [2]. What the colourings can be in a painting, and discovering how to make them, is as much a part of painting as discovering the colours and forms these colourings lead lookers to see. Painting is thus an exploration

of colourings and the forms they can make, and I paint what I discover, not something I'm looking for, nor have in my mind. My "vision for the model [the painting]" is always the painting as it currently is, not something already formed in my head. Exploration is trying things out to discover where they can take this vision. Painting stops when I discover a place I like.

Seeing is a natural consequence of some looking, usually: the rest state of a looking-to-seeing process. Mostly looking-to-seeing happens so fast we don't notice it. We don't experience it. This can lead us to think only seeing happens. But sometimes looking-to-seeing can be slowed down, so that we arrive at our seeing slowly. Occasionally it can even be stopped, so that no seeing happens at all: we can't say what we are looking at. I am interested in the looking-to-seeing process, and how it can be slowed down. Painting is a way to study this.

Investigating looking-to-seeing by painting has led me to regard looking as a kind of distal "touching," and seeing as like the experience of forms and impressions we get from touching, feeling, and caressing something with our hands and fingers, and sometimes more of our bodies. Stealing the term from work in Film Media, I call this *haptic visuality* [3]. It leads to an emphasis on the texture and the visual and emotional "feel" of colourings, and an avoidance of lines as denoting boarders and outlines.

Computational Painting

I use 'computational painting' in the same way that terms like 'acrylic painting,' 'oil painting,' 'spray painting,' and 'watercolour painting,' are used to name different ways of making paintings. Like these names, computational painting picks out the kind of stuff I use and work with to make paintings. Unlike these other, more conventional, kinds of painting, there is a separation of the stuff I use and work with and the colouring that is put on the painting surface. I don't apply computation to the canvas. I use inkjet printing to put different coloured inks on to paper. To make this

happen, I make a standard PDF file from the outcome of the computations. This inkjet printing happens at the end, when I think I have a finished painting. While doing the painting, what the current computer code produces is displayed on a colour screen — the LCD screen of the Apple PowerBook computer I mostly use while painting.

This way of painting comes with certain conveniences: it's cleaner and cheaper than more conventional painting practices, even including the cost of high quality inkjet printing. More importantly, it gives me a way of working and thinking that I like and feel comfortable with, compared to using brushes and paints, for example. Making and changing computer code, and seeing what I get from the computations of the executed code, gives a precise and finely adjustable way to control colouring, how it is made, and where and how it is put on the painting surface. This precision, together with the almost limitless possibilities of programed computation, and a convenient means to display its outcomes, makes it, for me, a practical way to make paintings, but not an easy way.

There are difficulties and complications in this way of painting, as there are, of course, in all other ways of painting. When you have near infinite easily modifiable possibilities, effective exploration, to discover interesting things, is not made easy. Limitations and constraints first need to be built to usefully discipline the exploration. This takes as much imagination and invention as does exploring the possibilities within this framing, and it always risks blocking access to exciting other possibilities. More. The colouring on the LCD computer screen is made of emitted red, green, and blue light combinations: the RGB colours. This is very different from the colourings made using the ten (typically) different ink colours printed on paper by the inkjet printer, which are, of course, reflected light colourings [4]. Understanding and working with this transition, from emitted RGB light colourings to the multi-tone ink colourings printed on paper, is a complication more conventional painting practices don't have to contend with.

Logo Programming

Today, there are a variety of programming languages suited to producing coloured graphical outputs. *Processing* is a good example, built for and widely used by Makers and Creators [5]. In 1985, when I first attempted to do this kind of computational painting, there were few such options, but there was Logo and its *Turtle Graphics* functions.

I first came across Logo when I moved to the Department of Artificial Intelligence (as it was called then), at the University of Edinburgh, from a background in aeronautical engineering, structural design, and lots of Fortran programming. The *AI and Education* group in the Department were using Logo, and developing Logo implementations for several of the then available Personal Computers (PCs). None of these PC systems had colour screens. I had a RGB colour monitor for my BBC Micro, but this was not one of the supported PCs. So I built a simple (and quite limited) Logo Turtle Graphics interpreter (in BBC Basic), to be able to make coloured Turtle graphics. It was slow, and I had no way of making printed copies of my simple computational paintings. This is how things stayed until about 2005, when I acquired a new Apple Mac computer running the (then) new OS X operating system, and when, more importantly, I found Alan C Smith's nice implementation of Logo for Mac OS X [6]. The other important development, since those early Edinburgh days, has been the arrival of high quality low cost inkjet printing. This has made it possible to turn the computational output into real paintings in an affordable and convenient way.

Logo is an early AI programming language in the LISP tradition, but with a much nicer syntax. It's better known *Turtle Graphics* was added later to support making Turtle drawings with what Papert called *body-syntonic reasoning*: understanding, reasoning about, and predicting the Turtle's drawing actions by imaging how you would need to move and act to draw what you want on the floor. This is what is behind Turtle actions like `Forward`, `Back`, `Right`

(turn), and `Left` (turn), etc. These are local actions, free of any coordinate frame, that allow you to drive the Turtle around the screen — the painting surface — getting it to put down colouring as it goes, like the way a hand moves a paint brush over a canvas, leaving colouring as it moves. This is quite different from most other graphical programming languages which are more geometric in their functionality, making it easy to specify the positions of start and end points and the kind of line that connects them. This is not what you do when you paint. The start and end of a painting stroke are only defined after the stroke has been made, and some colouring has been put down. You might be careful about where you start a stroke, but where it finishes is where it turns out it needs to finish as you make the stroke. This is an easy and natural thing to do in Logo, and is how I build what I call *computational paintbrushes*. This is an important way to introduce need framing constraints and limitations. Having made a paintbrush, I explore what can be done with it. A nice thing about computational paintbrushes is that they can be made to put down multiple colourings in one stroke, unlike in oil or watercolour painting, or indeed any other way of applying real paint to a painting surface. Another nice thing is that computational brush strokes can be as long as you like, from very short to very very long.

Making the Seymour Papert Painting

Wanting to paint something to celebrate and remember Seymour Papert doesn't get any painting started. To do that I needed something to try. For this painting two things came to mind. One was the paradigmatically Logo figures of the regular polygons — the triangle, square, pentagon, etc — the things all (it seems) Logo learners first learn to make. The other was something I did for a painting of Salvador Dali I made a few years ago. This started with a sampling of a Black and White photograph of Dali (which is, of course, a grey scale image, not just black and white). I already had a Black and White photograph of Papert which I like for his quiet playful smile.

I worked on these two ideas separately, but at the same time, sometimes working on one, sometimes the other. Initially I thought one of them (or none of them) might lead to a painting, but I saw something in a combination of polygons I had discovered that suggested a way to combine this idea with the sampled photograph idea. This is not how I usually work on a painting, working on more than one idea at a time. This time, working on both ideas formed a kind joint construal for what become the one final painting.

Drawing regular polygons doesn't easily result in something interesting, so I built in relationships between the lengths of their sides and the locations of their starting points, and explored what happened as I changed these. In other words, I built a *construal* to work with. This is typically what happens when I add constraints and limitations in an attempt to make some exploration doable. When this construal making works well, the emptiness that a near infinite number of possibilities presents becomes a detailed and promising terrain to explore and extend.

The polygon combination I discovered, and liked, is composed of eleven *Prime Polygons*: polygons with their number of sides equal to the first eleven Prime Numbers, starting with 2, which forms a line, and which I (perhaps unconventionally) take to be a two-sided polygon. The length of the sides of each polygon, starting with the 2-polygon as the base length, is reduced by the reciprocal of the Golden Ratio (0.618...) — a magic number I like. And each polygon is evenly shifted rightwards by a constant amount, scaled to make the complete figure fit well in the frame of the painting. This results in a combination of geometric forms easily drawn with a Logo Turtle, but which displays several nice properties and puzzles, which I think Seymour would have liked. For example, the left-hand (verticle) sides of the polygons are evenly spaced, because that's how they are painted, but their right-most corners (which they have because they are odd number sided polygons, except for the first one) are not simply arranged: why? It is not just a result of the scaling of the sizes of the polygons.

Some years ago, Alan Smith, the builder of the ACSLogo I use, kindly added a function to sample the RGB colour values at the current Turtle position. (I asked for this to have a way of modifying the colouring put down by the Turtle according to the colouring already there.) ACSLogo also has a function to load in, position, and scale, an image file, such as a jpeg photograph. In combination, these two Turtle functions provide a way to load in and sample an existing image, such as a photograph. This needs the Turtle to be moved to enough different points to usefully sample the different RGB values of the original image. A simple way to generate a distribution of these sampling points is to use the random number function — to make random coordinate pairs — but I generally don't like using random numbers in my paintings. Instead I use a Logistic Map function set to operate in its fully chaotic mode, and seeded differently each time it is run with a number derived from the system clock at execution time. With some adjustment to the coordinates made this way, to move them away from the edges some, this gives me a way to sample the RGB colourings of a loaded photograph, which I then save to a file, together with the corresponding sample point positions. By varying the total number of sample points, I can easily change the "resolution" of the saved sample set.

I used these stored data to paint points on a clean canvas (without the original photograph) using colourings for each point made using the original Black and White RBG values combined with a new colouring mechanism. This makes it possible to paint images that recreate the photograph image at a lower and more sparse resolution with different colourings. In this painting I also decided to use a kind of point I have used and liked before: a circular point with a hole in its centre, to let some colouring through from behind. This kind of "pointillist" painting can give a nice texture to the colouring: a haptic visuality which, close up gives a *sequined* look to the colourings, but which, further away, lets an image of Papert emerge, still with his playful smile. After exploring the effects of different numbers of sample points, and different sizes of painted

points, I settled upon using one million colouring points, all of the same size.

At the scale it is draw, the 11th Prime Polygon, with 31 sides, looks almost circular, and with a diameter close to the size of Papert's left eye, in the pointillist image I had made from the photograph. So I decided to combine both these ideas, to have Papert looking through the 13th Prime Polygon, which just needed a little scaling and adjustment of the positioning of the set of polygons. I didn't like the plane colourings of the Turtle drawn polygons when I first put these two schemes together, so I used computational paint brush I had made before to paint the lines of the polygon sides, to give them some texturing too. If you look carefully, you can see this in the painting.

I thought the painting was finished, but some days after stopping I looked at it again and saw that it needed a Turtle. I tried several different ways of painting a turtle, but settled upon using the same sampling and re-painting technique I had used for image of Papert. I drew, on paper, a turtle, copying an image I found on the web, sampled a scaled up photographed of this, and then use the sampled point set to paint an all-coloured Turtle, again using points with holes in them. The place to put this Turtle naturally seemed to be at the start of the triangle that everybody makes when they first start to learn to program in Logo: at the start of the green triangle that connects the red line (of the 2-polygon) to the smiling face of Seymour Papert looking through the 13-sided Prime Polygon.

With this addition of a Turtle I decide the painting was finished. And, as often happens, in a way I'm hardly aware of, a title for this painting came to me: *Painting With a Turtle to See More*. I like playing with words as much as I like playing with computational paints. [7].

And the Program?

Occasionally people ask to see the program that makes the painting, some adding that this too is part of the created

work. I reply saying I have no program to show, that I make paintings not computer programs.

This surprises people, and is, I think, different to how others work. A computer program has a known purpose, and it needs to be designed and engineering to fulfill this purpose, in a (known to be) correct, effective, efficient, reliable, robust, maintainable, and, or course, usable way. I have designed and built many programs like this, or worked with others to do this; programs whose correct and efficient working other people depended upon. When I start painting I don't have any idea of how it will turn out, nor what it should be. It is therefore not possible to design and build a program to make the painting. What I do do is build lumps of Logo code that do things, and which I then work with. I make *construals*, some of which, usually after much modification and changing, form the Logo code used to generate the PDF from which a painting is made. Sometimes this Logo code does look like a program, but on other occasions, the painting is built by running different lumps of code. This is the intended way of working with Logo: make something, see what it does, do something else that you now see could be tried, and keep doing this. This is making, thinking, exploring, discovering with Logo code, but it is not, I would say, program building, not in the good engineering sense that I think program building should be done.

I am careful about the code I build, but not particularly neat and tidy about how it is organised. The code does need to be correct in the sense that I am sure I understand how and why it produces the colourings and forms it does. I have discovered interesting, sometimes striking, things from buggy code, but to use these discoveries I have to build correct code that also produces them. The Logo code I build and work with, while working on a painting, is perhaps best thought of as being more like the paints, (mixing) pallet, and brushes of the oil painter. These are not viewed by anybody as being a part of the final painting, but they are essential to its making, and, just as importantly, they

are and integral part of the thinking and working out the painter had to do to arrive at the final painting.

Notes

1 Construit! The European Erasmus+ Project on Empirical Modelling for Education. (⟵)

2 We typically see trees as green, and say trees are green, but a careful look shows the leaves of any tree to be many different greens and other (not-green) colours too, and even no greens, when seen in the light of a yellow streetlamp, for example. See Mazvitta Chirimuuta, 2015: *Outside Color, Perceptual Science and the Puzzle of Color in Philosophy*, MIT Press, for an interesting treatment of what is colour. (Though one, like others, that stuffers for not using a distinction between colourings and colour, I would say.) (⟵)

3 I first came across the idea of *Haptic Visuality* in the work of Laura U Marks. I stumbled upon *Touching the Film Object?*, a short video by Catherine Grant which explores Laura Mark's notion of hapticity in a visual medium. In this video, Catherine Grant refers to the art historian Alöis Riegl's "distinction between haptic and optical images," as Marks describes it. Riegl was, amongst many other forms of art, interested in Persian carpets with their endless interleaved patterns. Riegl said (quoted by Grant) that these ...

> *"... patterns don't allow the eye to rest in one place; they invite the eye to move along them, caressing their surface. Contemplating these patterns does something to dissolve the boundaries between the beholder and the thing beheld."*

This "inviting of the eye" to keep moving over the looked at pattern, and this "dissolving of the boundaries" between the subject doing the looking and the object seen, captures well what I think happens when slow looking-to-seeing "reaches out to touch" the visual scene. The reaching out and touching brings things so close to the looking that you can't see a "big picture," the *optical*, in Riegl's terms, but you can see the details, the *haptic*, says Riegl, and these

details drive the looking, the visually haptic touching, of the surface. What is depth in more distanced fast seeing becomes texture in this slowed down up close seeing: texture that invites the eye to caress its surface.

Riegl wasn't the only, nor first, to talk like this. Much earlier Johann Gottfried Herder said, when talking about how we look at sculptures, as opposed to (flat) paintings:

> *"The eye that gathers impressions is no longer the eye that sees a depiction on a surface; it becomes a hand, the ray of light becomes a finger, and the imagination becomes a form of immediate touching."*

See Johann Gottfried Herder, 1778. *Sculpture: Some Observations on Shape and Form from Pygmalion's Creative Dreams*, translated by Jason Gaiger, Chicago University Press, 2002, pp 19. (⟷)

4 The ten Epson inkjet colours I use are vivid magenta, yellow, cyan, orange, green, vivid light magenta, light cyan, light black, matte black or photo black, plus an irreversible choice of either light light gray or violet. (⟷)

5 Casey Reas and Ben Fry, 2014: *Processing, A Programming Handbook for Visual Designers and Artists*, MIT Press. (⟷)

6 ACSLogo for Mac OS X. (⟷)

7 Words and languaging, are very closely tied up with seeing. If you cannot say what you are looking at, are you seeing anything? If you think you are seeing something, but can't say anything about what it is you're seeing, how do you know you are seeing anything? For more paintings by Afriko, click on these words. (⟷)