← {Software} Structures

A text about Software & Art by Casey Reas

Software & Drawing

I want programming to be as immediate and fluid as drawing and I work with software in a way that minimizes the technical aspects. I often spend a few days creating a core piece of technical code and then months working with it intuitively, modifying it without considering the core algorithms. I use the same code base to create myriad variations as I operate on the fundamental code structure as if it were a drawing - erasing, redrawing, reshaping lines, molding the surface through instinctual actions. In the past year, I have begun removing code from the process of creation. The concept for the work develops entirely through sketches and the final piece is an annotated written description without reference to a computational implementation. The work develops in the vague domain of image and then matures in the more defined structures of natural language before any thought is given to a specific machine implementation. I'm calling this type of program a software structure.

A defining factor in this shift was the work of Sol LeWitt - specifically his wall drawings. A wall drawing is a set of instructions (a text description and optional diagram) outlining a visual structure to be executed on a wall. For example, the program for Wall Drawing #69 from 1971 reads:

> *Lines not long, not strait, not touching, drawn at random using four colors, uniformly dispersed with maximum density, covering the entire surface of the wall.*

LeWitt has written hundreds of wall drawings since their origin in 1968. Each time a wall drawing is reproduced it is different depending on the site and the draftsperson. There is a complete separation of the concept of the work from its perceptual manifestation. The relation between LeWitt and his draftsperson is often compared to the relation between a *composer* and *performer* , but I think it's also valid to look at the comparison between a *programmer* and the *entity* of execution. LeWitt writes programs for people to execute and interpret rather than for machines. This difference allows him to work in natural language, rather than the limited formal languages of computer code. He is also able to leave ambiguity in his programs, as they will be executed by skilled draftspersons who are able to interpret, rather than a machine which must be told precisely what to do.

This essay explores the potential influences of LeWitt's work on contemporary works of software. I have created three example software structures. To further define the concept of software structures I have written and commissioned twenty-four implementations of these structures to isolate different aspects including interpretation, material, and process.

UP

Sol & Software

There are a few obvious approaches to analyzing LeWitt in relation to software: implementing his work as programs; re-coding his concepts using different notation systems; and creating generative structures encoding his methods. I'm more interested in less obvious and transformational relations, including separating the concept of the work from the implementation, pursuing intuition, and the potential for dynamic structures.

Separating Concept & Implementation

LeWitt continually emphasizes the separation between conceiving a drawing and its realization. He promotes his role as the conceptual artist and underplays the role of the draftsperson by writing statements such as, "What the work of art looks like isn't too important." [1] But he is actually concerned about the final perception of the work as he later wrote the plan "needs to be put in optimum form," [2] "the same tonality should be maintained throughout the entire plane," and "the best surface to draw on is plaster." [3] This is not a strong contradiction as he allows for many variations on his plan and is only concerned with lack of consistency and poor quality not conveying the structure.

As an artist working with software, I'm fascinated by LeWitt's decision to filter his ideas through an intermediate text representation and through the preferences of an intermediate person. It provides the freedom to work at a more abstract level and to experience more variations on the core concept of the work. The decision to use natural language makes the work more open for interpretation by the draftsperson and also more accessible for the audience.

Wall & Screens

Each wall drawing adapts itself to the site, meaning, each wall's unique surface and proportions yields a different drawing. Lewitt wrote, "The physical properties of the wall: height, length, color, material, and architectural conditions and intrusions, are a necessary part of the wall drawings." [4] He considers the differences and imperfections in each wall as an element of the drawing, but the structure he predefines dominates the perception. The information remains intact despite different levels of noise in the signal. In terms of software, analogies may be made between the wall and the screen (different sizes, resolution) and the wall and the hardware on which the software is running (different speeds and operating systems). Projecting software on a wall creates more potential similarities between wall drawings and software structures.

The Mystic Programmer

The first three statements in LeWitt's 1969 text "Sentences on Conceptual Art" read:

[5]

> *1. Conceptual Artists are mystics rather than rationalists. They leap to conclusions that logic cannot reach*
> *2. Rational judgements repeat rational judgements.*
> *3. Irrational judgements lead to new experience.*

LeWitt continually emphasizes these sentiments in his writings by consistently using the terms *illogical* and *intuitive* . While these terms easily categorize human thoughts, they cannot be as easily used to define software. Many pieces of software may have their origins in a quick impulsive decision, but as soon as they are made manifest in code they become rigid and fixed as dictated by the constraints of the technology. If the final program was specified through a combination of diagrams and text descriptions, it would be possible for the artist to work a longer time and more often in an undefined mental space.

Dynamic Structure

LeWitt has spent his entire career to date planning wall drawings with static structures. Drawing on walls constrains form to a specific place and therefore maintains the same relationships between elements as time passes. In software, it is possible to create structures defining dynamic relationships between elements. In a wall drawing, the plan may declare "draw a line" but in a software structure the plan may declare "draw a line moving from left to right." In a wall, drawing the plan may declare "elements not touching" but in a software structure the plan may declare "when two elements touch, create a new line." Elements in a software structure can move and have behaviors in relation to the other elements, while wall drawings have dynamic relations in the mind of the artists and draftspersons, but remain static after the drawing is completed.

UP

Implementations

To explore the potential differences and similarities between software and LeWitt's techniques, three of his wall drawings were implemented and extended as software.

Wall Drawing #85

The plan for Wall Drawing #85 defines four quadrants, each filled with different patterns of colored lines:

> *85. Same as 63, but with four colors.*
> *63. A Wall is divided into four horizontal parts. In the top row are four equal divisions, each with lines in a different direction. In the second row, six double combinations; in the third row, four triple combinations; in the bottom row, all four combinations superimposed.*

Implementing this description in software was an exercise in translation, simply converting the instructions into a format the computer will understand. Through this act, the decisions intended for the draftsperson were made by myself, the programmer. A few obvious differences appear as a result of this re-coding. Computer screens have a much coarser resolution than a wall and as a result, the finished work lacks the warmth of a drawn surface in a physical space. In addition, machines can draw lines with absolute precision so all the *imperfections* in a physical drawing are removed, giving the rendering different characteristics than those intended by LeWitt. Do these differences distort the result? If this is a work of conceptual art, the concept should remain regardless of the medium.

Wall Drawing #106

The plan for Wall Drawing #106 defines a series of arcs drawn from the edge of the wall:

> *Arcs from the midpoints of two sides of the wall.*

Executing Wall Drawing #106 in software brings into focus the coarse resolution of the screen as it creates many unintended artifacts. Another noticeable difference is the scale of the work. Executing a drawing in software means creating it without knowledge of the size of the output. A person could view this work on a small screen, an enormous display in an urban center, or could project it at the same scale as the original drawing on a wall. Flexibility in size is a part of the LeWitt's original wall drawings, but the potential difference is less.

In the original plan, LeWitt doesn't specify the number of arcs to be drawn, but leaves this decision to the draftsperson. I have modified this drawing so that moving the mouse left and right across the surface of the image changes the number of arcs that are drawn. When the mouse is at the far right, there is only one arc drawn from either side and when the mouse is at the far left, there are over five hundred. As the mouse moves left, the arcs visually collide, creating a moiré pattern. This is a technically trivial change introduced with one line of code, but I feel it's a radical re-invention of LeWitt's concept and reveals the potential for creating responsive drawings in a software environment.

Wall Drawing #358

The plan for Wall Drawing #358 defines a series of arcs drawn in a grid:

> *A 12" (30 cm) grid covering the wall. Within each 12" (30 cm) square, one arc from the corner. (The direction of the arcs and their placement are determined by the draftsman.)*

The most obvious difference between the original structure and the software modification was the translation of the measurements from inches into pixels. I chose to make a grid of 50 pixels, thus creating a matrix of 16 x 12 units. Wall Drawing #358 includes the explicit instruction for the draftspersons to make their own decisions. In the software implementation, the program displays a set of arcs selected at random. This removes the decision process from the draftsperson and relies on random selection. Does this degrade the result of the final composition? Is it necessary that the decision of how to orient arcs is made by a person?

The variation of Wall Drawing #358 continually updates itself four times each second, displaying a randomly selected composition. This expression of Wall Drawing #358 gives the viewer a better understanding of the structure behind the drawing than looking at a single instance of the system. It could have been interesting to build a version where the viewer has the ability to change the direction of the arcs, thus taking this decision away from the draftsperson and giving it to the viewer.

UP

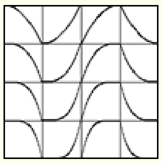
Structures

Without diverting into a didactic text about Structuralism, I will simply state that a structure is a relationship among elements. Within disciplines such as linguistics and anthropology, structures exist but are open to re-interpretation, but in computer science, structures are by necessity extremely precise. Computers are machines designed for their *reliability* and *accuracy* and while these characteristics make computers valuable to the scientific and engineering communities, they are not necessarily important characteristics for an artist. Artists such as Takashi Murakami use the computer as a precise tool, but it's not possible to imagine the work of Anselm Kiefer put through the filter of a precise computing machine.

Some structures are unique to software and are not possible to express in other media. Software is excellent at defining processes. In the words of MIT Professor Harold Abelson, "processes manipulate other abstract things called *data*. The evolution of a process is directed by a pattern of rules called a *program* . People create programs to direct processes." [6] The potential of defining systems in software is revealed through the following three structures.

Software Structure #001

Every possible pairing of these sixteen curves:



Use the additive numeric values from each curve to set the value of a series of horizontal lines from white to black.

Structure #001 uses one of LeWitt's common techniques, pairing a group of variables in every possible configuration. In this structure, the changes occur in time rather than space. Every curve in the diagram is paired with every other curve to determine the values of the horizontal bars. An oscillator moves continually along the x-axis of each curve with the corresponding y-axis values added to define the momentary gray value of each bar.

Software Structure #002

A grid of points in the top half of the surface. Each point moves downward and returns to the top when it leaves the bottom edge. Beginning in the upper-left, each row and column moves faster than the previous. The speeds combine so the point in the upper-left is the slowest and the point in the lower-right is the fastest. Copy and flip the grid across the central vertical axis.

The initial strong spatial configuration dissolves into a patterned field. Each column follows the same rhythm, but with a different phase. The surface is continually changing with macro-structures temporarily emerging and dissipating. In this structure, there is no coded relation from one element to another, but the elements appear similar because they all follow the same instructions at a different speed.

Software Structure #003

A surface filled with one hundred medium to small sized circles. Each circle has a different size and direction, but moves at the same slow rate. Display:
A. The instantaneous intersections of the circles
B. The aggregate intersections of the circles

A. The points moving on the screen are the center of each circle and the lines connect the intersections on the perimeters of overlapping circles. To emphasize the difference in line length, they increase in value from white to black as they grow longer. Because the edge of a circle is a nonlinear surface, the lines drawn from the intersections begin to grow quickly, linger as they approach maximum length, and diminish quickly. When three or more circles overlap, the illusion of depth increases as the resulting lines construct a corner.

B. The aggregate version is like photographing the instantaneous version with the shutter left open for an extended time. It gives a different vision into the structure by compressing changes over time into the same visual space. The transition of the line values from white to black is critical to avoid that the screen saturates entirely with one value. The image created by this software is continually changing and never repeating.

UP

Anatomy of a Software Structure

Many decisions are made while implementing a software structure. They include visual decisions about form and motion, technical implementation issues, such as the language and algorithms to use, and the process in which the structure will develop. To isolate these variables and make the decisions transparent, three people (Jared Tarbell, Robert Hodgin, and William Ngan) were commissioned to implement the same structure; the structure was implemented in three separate programming languages (Processing, ActionScript, C++); and the process of realizing one structure was revealed from the first lines of code to the completed work.

Interpretation

When a piece of music is performed, the musician interprets the composer's score according to her own ideas and in relation to the composer's intent. This relation became more pronounced in the 1960s through the work of composers like Karlheinz Stockhausen. Umberto Eco wrote in 1962: [7]

> *A number of recent pieces of instrumental music are linked by a common feature: the considerable autonomy left to the individual performer in the way he chooses to play the work. Thus, he is not merely free to interpret the composer's instructions following his own discretion (which in fact happens in traditional music), but he must impose his judgement on the form of the piece.*

When Sol LeWitt introduced his systemic drawings, a similar relation entered the visual arts. In "Doing Wall Drawings" LeWitt wrote, "The draftsman perceives the artist's plan, then reorders it to his experience and understanding" and "The artist and the draftsman become collaborators in making the art." [8] The degree of interpretive freedom an artist gives to the person executing the work is specific to each piece. Regardless, there is a deep structure that reveals the intention of the author. For this project, three individuals implemented Software Structure #003 without communication or knowledge of the others' direction and without knowledge of the original implementation.

Jared Tarbell

A. Each circle begins as a point and grows to sizes between ten and fifty pixels. Circles are drawn as points moving around a perimeter and when two circles intersect, a larger white dot marks this place. An irregular temporal pattern is created as the elements intersect and break apart.

B. The circles in the aggregate version of the structure draw variegated lines between their points of intersection and Jared intentionally simplified the movement of the circles to not obscure the underlying structure. It is fascinating to compare this implementation to the original as many similar forms are revealed through the circles' interactions.

Robert Hodgin

A. Each circle continually monitors its distance from the others. If the distance between two circles is less than a predefined number, they connect and move together, but if an intersection occurs, they disconnect and are repelled. A weak force slowly moves them toward the center.

The aggregate version presents a visualization reminiscent of Jared's first software and with a theme similar to Robert's first interpretation. The elements begin in a circular configuration, a weak force moves them toward the center, but their collisions propel them outward. Collisions are displayed with a flash of white when they are outside the original circle and a flash of black when they are inside.

Robert based the actions on collisions and not points of intersection and developed forces which act upon the circles, rather than giving them constant linear motion. He also had a different idea of the phrase "medium to small" in relation to the size of the screen. These divergences from the original intent make the interpretations extremely interesting and surprisingly varied.

William Ngan

A. William's intent was to give the circles a sense of life. They slowly drift across the screen and upon intersection, they orient themselves to the adjacent circle. When an intersection is brief or there are simultaneous crossings, it's easy to feel the circles are *indecisive* or *nervous.*

B. The aggregate version implies the circles as they deform a field of lines. They begin in the center and slowly disperse. When two circles intersect, the lines overlaying each circle orient to the position of intersection. The result is an organic field of structured waves.

Material

Artists use a wide range of materials to great effect: leather, honey, blood, oil, steel, felt, latex, paper, rubber, plastic, bones, cotton, concrete, glass, ceramics, copper, etc. The choice of material affects the perception of the work and therefore a careful choice is critical to success. Artists working with the software medium also use a wide range of materials: Java, C++, Perl, PHP, BASIC, LISP, PostScript, Python, etc. These software materials are not as familiar to most people as the physical materials mentioned above, but regardless, the choice of programming language greatly affects the perception of a piece of software. Some programming materials allow working quickly, some require intense attention to detail, and all modify the way the programmer thinks about the structure. For this project, one structure was implemented in three different software materials to isolate the similarities and differences between each.

Processing (2001)

Processing was designed for the context of dynamic visual work. It has a carefully designed graphics library for the construction of 2D/3D visual form and color. Processing is written with Java and therefore draws more slowly than Flash and calculates slower than C++. Processing is free and open-source and is therefore extremely accessible. Processing was used as the primary language for the project because it is easily viewable over the Web and the code is fast to write and easy to read.

Flash MX (1996)

Flash is an environment originally developed for efficient Web animation and it has been improving its programming functionality since 1998. It draws much faster than Processing, but calculates much slower. Flash was built for creating 2D graphics and is able to render flat graphics and typography of an extremely high visual quality.

Software with hundreds of elements all performing intense calculation (e.g. Software Structure #003) runs so poorly in Flash that the intent of the work is destroyed. A different structure with intense drawing and without excessive calculation would run excellently in Flash.

C++/OpenGL (1979/1992)

C has been an extremely popular language for computer scientists since its origin in 1970. C++ is an object-oriented version developed since 1979. OpenGL is a graphics library originally developed as IRIS GL by Silicon Graphics. Using an OpenGL accelerated graphics card in a computer (the kind used by video game enthusiasts) allows an outstanding resolution and speed in comparison to Processing and Flash. C++ programs using OpenGL are not able to run over the Internet. The speed of C++ gives the software structures a fidelity which far surpasses the structures in Processing and Flash.

Process

Many of us have seen the romantic photographs of Jackson Pollock hunched over the canvas at his studio with a paint can in one hand and a brush in the other. He intently moves across the floor while dripping paint on the canvas as a painting slowly emerges. Through this type of romanticized media imagery, people have glimpsed the process of creating paintings, sculptures, and other traditional arts. The process of creating software is a mysterious practice with few references in popular culture and art discourse. Software is a very fragile material and working within its rigid syntax and structural rules can be very tedious for people who love to directly engage with physical materials. Writing software is a process of translating fuzzy ideas from one's mind into a strict notational system. Using this notation as an intermediate step, visual and kinetic ideas manifest themselves in computational machines. As with other types of art, software may be written through a process of intuitive exploration, and it can be written precisely to meet a goal.

I have chosen to reveal a series of steps in the implementation of Software Structure #003. While each successive piece is an increasingly complex technical study, the form of the final work slowly reveals itself and as in most evolutionary processes, it typically progresses in quick leaps rather than a continual incline. There were hundreds of separate versions of this software as it progressed over a period of two weeks. For clarity, ten representative steps have been selected to show the software's development.

01. The first piece of code detects the intersection between two circles. One circle is stationary and the other is controlled by the mouse.

02. Automating the position of the circles, thus removing any input from outside the running software.

03. The visualization has been modified, so that the circles are not being drawn and only their centers are displayed with a black pixel. Intersections are shown by a connecting line between the two points.

04. The circles are now moving diagonally and bouncing off the screen edges. The intersection is visualized by a line connecting the two points.

05. Only the line showing the intersections is displayed, and this line accumulates continuously into the screen buffer.

06. Places the correct number and size of circles on the screen. The circles are visualized clearly and not moving.

07. The circles are moving along diagonal paths, moving past the edge of the frame and returning to the image area on the opposite side.

08. The intersections are again visualized by lines and the size of the circles is doubled.

09. Only the linear intersections are displayed, thus implying the structure, rather than presenting a literal display.

10. The screen is not refreshing every frame, and the lines now accumulate in the image surface. Because only black lines are drawn, the screen continually darkens. The final version of the software is different only in the change of the lines' value from white (when they are short) to black (as they grow).

UP

Paragraphs on Software Art

Before starting this project, I had little idea what it would yield. After completion, it has created a foundation for future exploration through producing a series of engaging ideas that relate conceptual art to software.

A significant difference between LeWitt's structures and those I have described is the variation within interpretations. If the artist's intent is to have a high degree of consistency within each implementation, the details of the description must be more precise than the text created for Software Structures #001 - #003. Additional clear description of the qualities of motion and the specific interactions between elements will communicate the intent more accurately.

A benefit of working with software structures instead of programming languages is that it places the work outside the current technological framework, which is continually becoming obsolete. Because a software structure is independent from a specific technology, it is possible to continually create manifestations of any software structure with current technology to avoid retrograde associations.

References

1. LeWitt, Sol. "Paragraphs on Conceptual Art," *Artforum* , 5:10. Summer 1967, pp.79-84. Reprinted in Alexander Alberro and Blake Stimson ed., *Conceptual Art: A Critical Anthology* (MIT Press, Cambridge, 1999) pp. 12-16.

2. LeWitt, Sol. "Doing Wall Drawings," *Art Now* . June 1971. Reprinted in Michael Auping, ed., *Drawing Rooms* (Modern Art Museum of Fort Worth, 1994), p.93,93.

3. LeWitt, Sol. "Wall Drawings." In Auping, ed., *Drawing Rooms* , p. 91.

4. LeWitt, Sol. "Wall Drawings," p.91.

5. LeWitt, Sol. "Sentences on Conceptual Art." 0-9, no. 5. January 1969, pp. 3-5. Reprinted in Alberro, ed., Conceptual Art, pp. 106-108.

6. Abelson, Harold, Gerald Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs* . MIT Press, Cambridge, MA. 1985. p.1.

7. Eco, Umberto. Translated by Anna Cancogni. The Open Work. Harvard University Press, Cambridge. 1989. p.1.

8. LeWitt, Sol. "Doing Wall Drawings." p.93.