

Process/Drawing

Writing software is at the core of Casey Reas's artistic practice. The digital is his medium of choice rather than a means of manipulation. He reflects on the evolution of his work in software and why the history of using computers to produce visual images is largely an unrecorded one in the history of art, but why this might all be set to change as scripting takes on a new primacy in contemporary art.

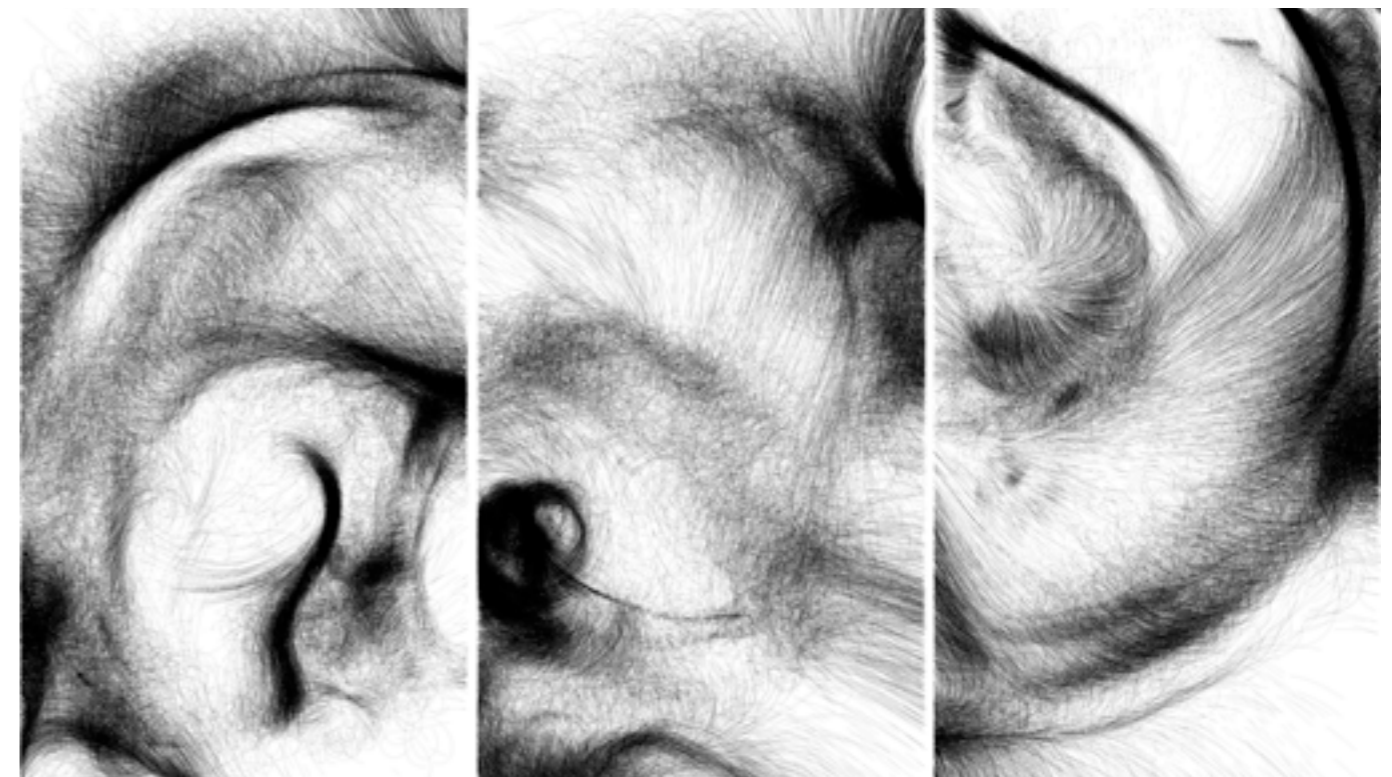
I started playing with computers as a child. Our family's Apple IIe machine was a toy for playing video games and writing simple programs in BASIC and Logo.¹ I spent years exploring and testing it, but I preferred drawing and my interest in computers slowly dissipated. In 1997 I was introduced to John Maeda and the work of his students in the Aesthetics and Computation Group at MIT. They were creating a unique software by fusing traditional arts knowledge with ideas from computer science. Experiencing their explorations revealed a new direction for my work and I started programming computers in earnest in 1998. I began my graduate studies at MIT the following year. My time there was personally transforming and I shifted from a software consumer to a producer. I expanded my views of technology in relation to culture and the history of art, and my current ideas emanate from this experience.

Writing software is now the core of my artistic practice. Many artists use software as a tool to support their work, but few are using it as a unique medium. It is now very common for photographers to manipulate their images digitally, for

painters to construct sketches and preparatory drawings with software, and for video artists to edit their work on computers. However, it is far less common for artists like myself to write their own software and to run and operate this software for their performances and installations.

For the past four years, my principle interest in software has been in exploring the phenomena of emergence. Emergence refers to the generation of structures that are not directly defined or controlled. Instead of overtly determining the entire structure, I write simple programs that define interactions between elements. Structure emerges from the discreet movements of each element as it modifies itself in relation to its environment. The structures generated through these processes cannot be anticipated, and develop through continual iterations involving alterations to the programs and exploring changes through interacting with the software. The works *Tissue* (2002) and *MicroImage* (2003) are among the results of this exploration.

In 2003 I became obsessed with the work of Sol LeWitt, and exploring his work had a clear impact on my recent software.

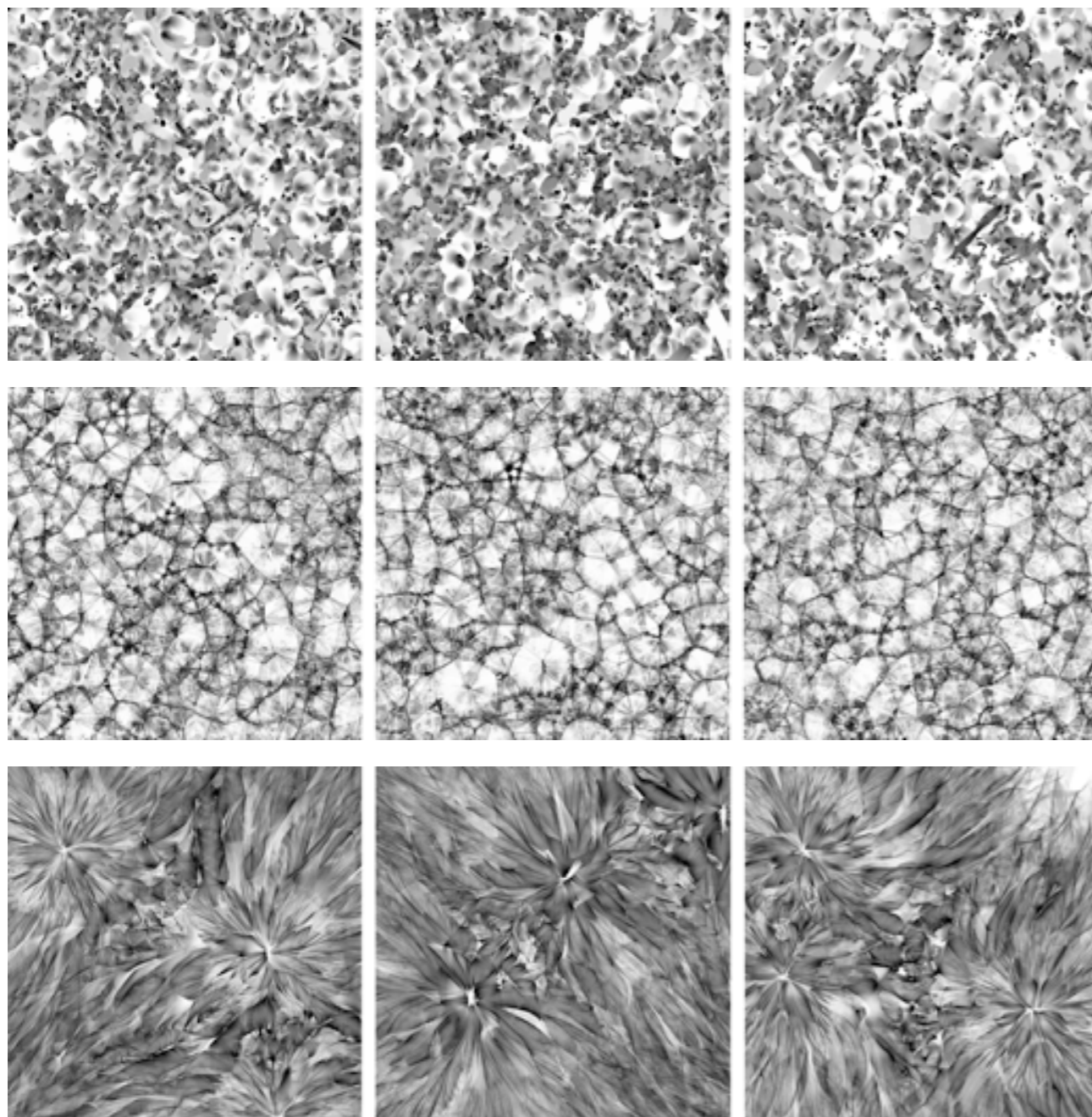


MicroImage, 24"x34" inkjet prints, 2003

Above: Three prints derived from the *MicroImage* software. The *MicroImage* software is an exploration into emergent form. Autonomous software elements interact with their continually changing environment to create a kinetic field.

Tissue Type B-06, 11"x14" inkjet print, 2002

Opposite: One of 28 prints derived from the *Tissue* software. The *Tissue* software exposes the movements of synthetic neural systems. People interact with the software by positioning a group of points on the screen. An understanding of the total system emerges from the relations between the positional input and the visual output.



Process 8 (Software), 2005

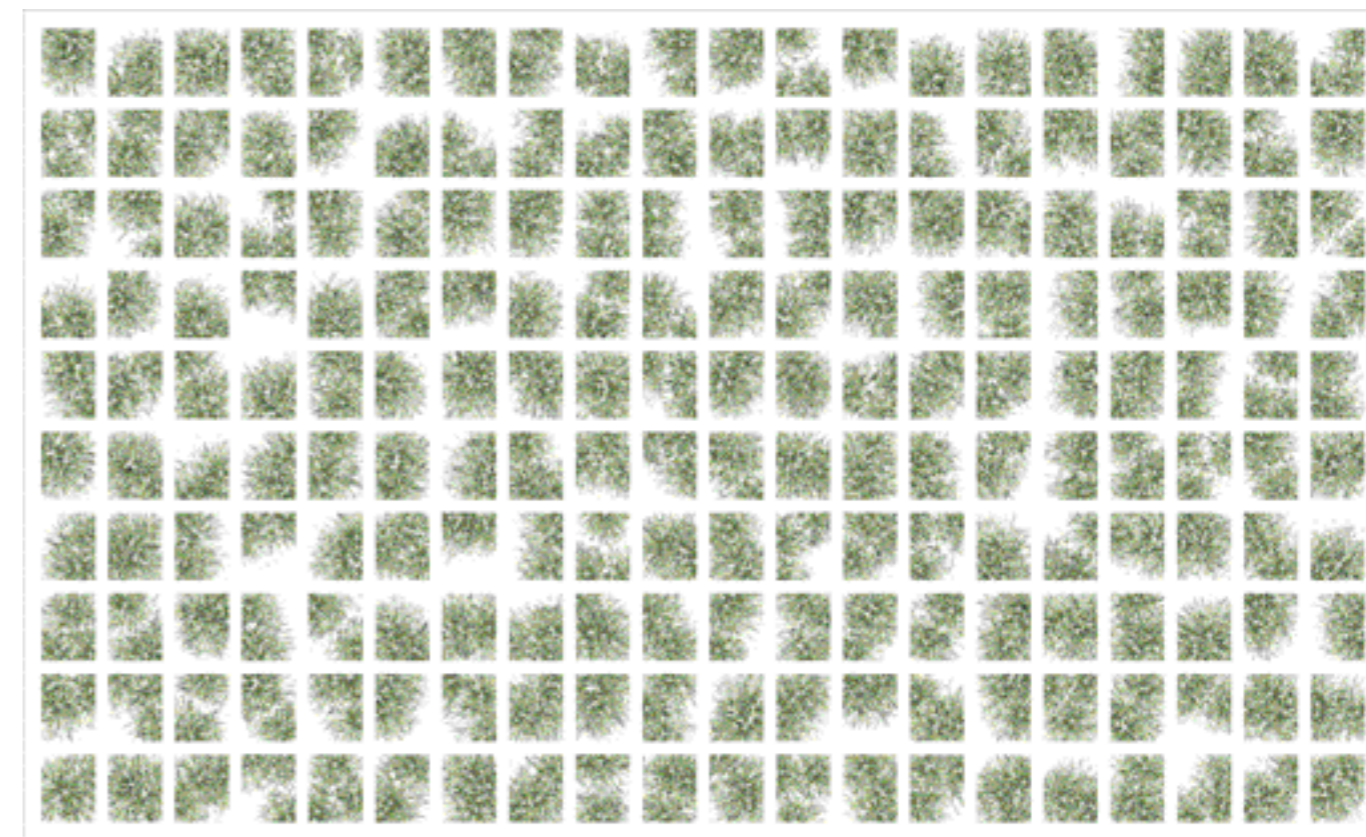
Top: Triptych of images from Process 8 (Software). This software was written from Process 8: A rectangular surface densely filled with instances of Element 2, each with a different size and speed. Display the intersections by drawing a circle at each point of contact. Set the size of each circle relative to the distance between the centres of the overlapping Elements. Draw the smallest possible circle as black and largest as white, with varying greys between.

Process 7 (Software), 2005

Middle: Triptych of images from Process 7 (Software). This software was written from Process 7: A rectangular surface filled with varying sizes of Element 1. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to black and the longest to white, with varying greys between. Draw the perimeter of each Element as a white line and the centre as a black dot.

Process 6 (Software), 2005

Bottom: Triptych of images from Process 6 (Software). This software was written from Process 6: Position three large circles on a rectangular surface. Set the centre of each circle as the origin for a large group of Element 1. When each Element moves beyond the edge of the circle, move its position back to the origin. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to white and the longest to black, with varying greys between.



Process 6 (Image 2), 36" x 22" inkjet print, 2005

Print derived from Process 6. Each grid unit is the same software run 400 iterations. Each unit was given a different initial position to create the visual diversity.

In spring 2004 I completed the Software {Structures}² project for the Whitney Museum of American Art's Artport. I started with a simple question: 'Is the history of conceptual art relevant to the idea of software as art?' I began to answer the question by implementing three of LeWitt's drawings in software and then making modifications. After working with the LeWitt plans, I created three new structures unique to software. These software structures are text descriptions outlining dynamic relations between elements. They develop in the vague domain of thought and then mature in the more defined structures of human language before any consideration is given to a specific machine implementation. Twenty-six pieces of software derived from these structures were written to isolate different components of software structures including interpretation, material and process. My work in the past year has synthesised ideas explored in Software {Structures} with my previous explorations of emergence. Each new work is called a 'Process' and defines rules and instructions that describe a relation between 'Elements' outside of a specific physical media. Different manifestations of the Process in diverse media such as software, print and installation give a more complete view of the concept. The software implementation is closest to the

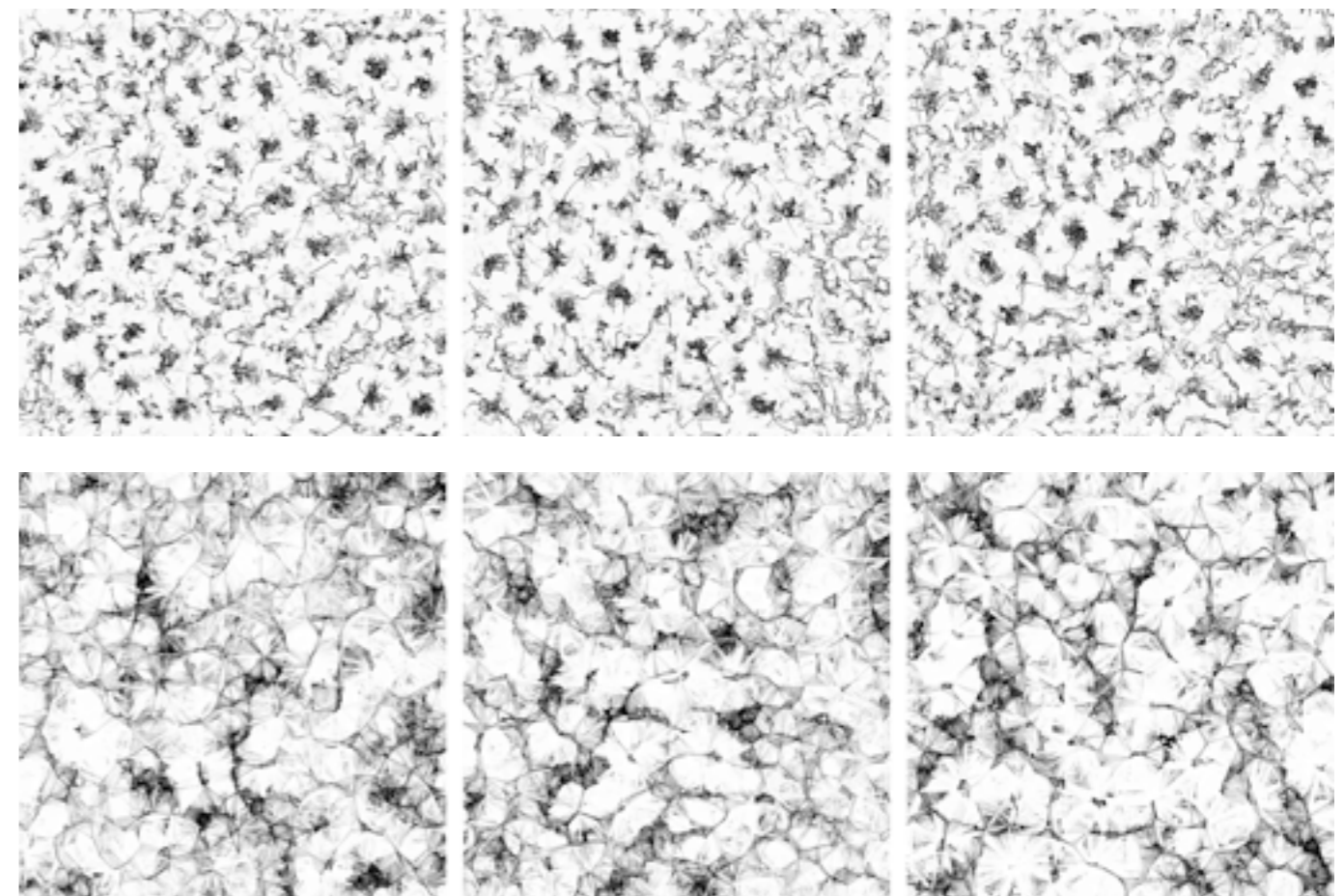
actual concept, but the other media provide additional perspectives. Working in software allows me to engage the live process and to change its parameters to affect the way it behaves, and working in print gives me the opportunity to examine the state of the process at a precise time with extremely high fidelity.

The most important element of each Process is the text. The text is the Process described in English, written with the intention of translating its content into software. Each Process defines the relationships of software Elements to their environment and to each other. Each Element is comprised of a visual form and one or more behaviours. Element 2, for example, is defined as Form 1 with Behaviours 1 and 4 applied. This means Form 1 is a circle with a constant linear motion, and when it moves off the edge of the screen, it moves to the opposing side of the screen. The remarkable aspect of each Process is the relationship between the simple instructions and the density and sensitivity of the images and movements they create. Each text is translated from natural language into a machine language to create the software. The Elements used in the Process texts are currently limited to two configurations, one form and four behaviours. This list will expand in time to increase the dynamic range of the



TI installation at the BANK Art Gallery, Los Angeles, 2004-05

Nineteen white discs hover above the floor, each containing a projected, kinetic image of expansion and decay. Aggregate layers of abstraction remove every trace of systemic complexity, revealing a living surface. Structured form emerges from the results of thousands of local interactions between autonomous elements.



Process 5 (Software), 2005

Top: Triptych of images from Process 5 (Software). This software was written from Process 5: A rectangular surface filled with varying sizes of Element 1. Draw the perimeter of each Element as a white line and the centre as a black dot. If two small Elements are touching, draw a grey line between their centres.

Process 4 (Software), 2005

Above: Triptych of images from Process 4 (Software.) This software was written from Process 4: A rectangular surface filled with varying sizes of Element 1. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to black and the longest to white, with varying greys between.

Processes:

Element 1: Form 1 + Behaviour 1 + Behaviour 2 + Behaviour 3

Element 2: Form 1 + Behaviour 1 + Behaviour 4

Form 1: Circle

Behaviour 1: Constant linear motion

Behaviour 2: Constrain to surface

Behaviour 3: When touching another, change direction

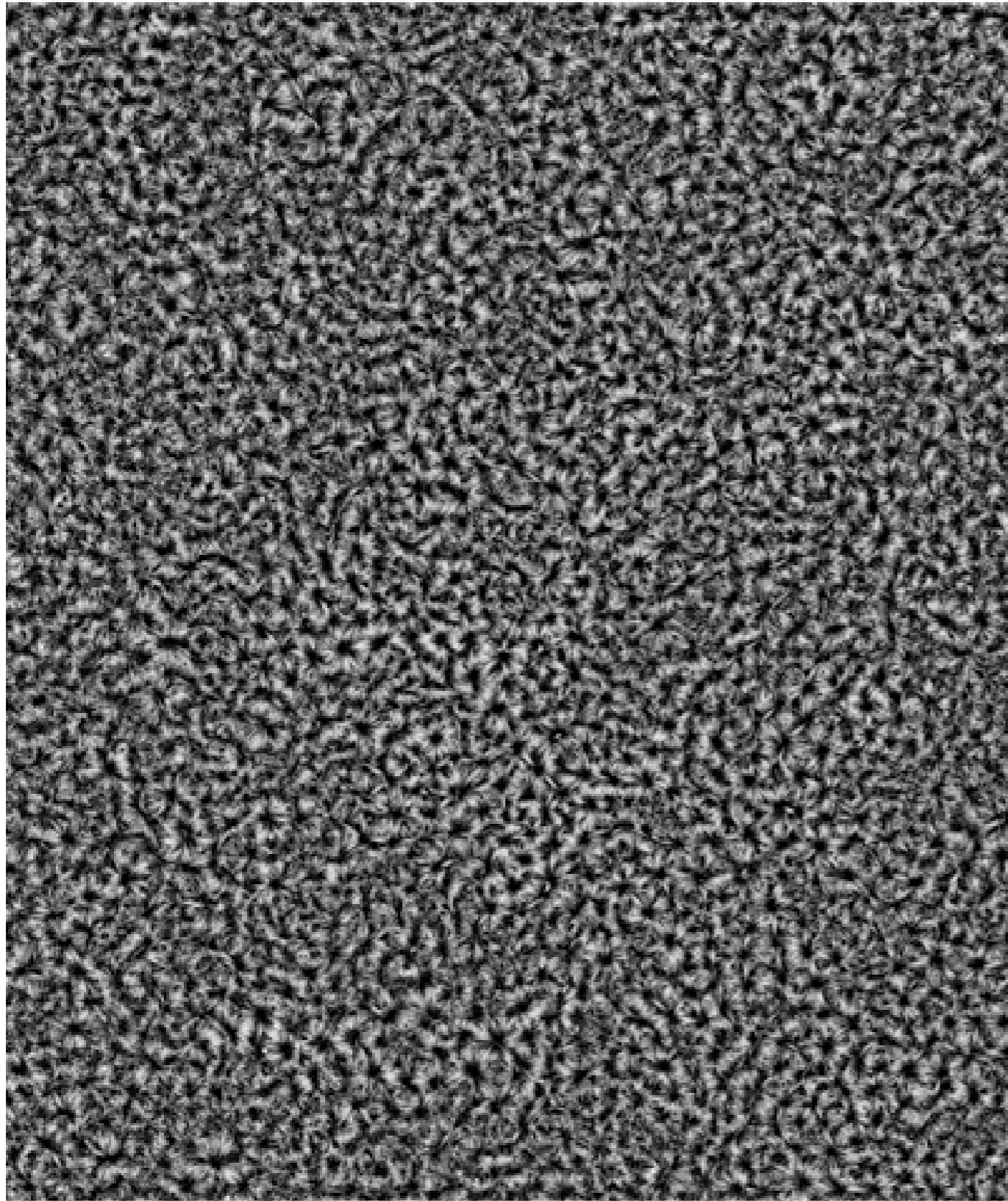
Behaviour 4: After moving off the surface, enter from the opposing edge

The texts for Processes 4 to 8 are reproduced below. Processes 1 to 3 were defined during the Software {Structures} project and additional Processes are continuously in development.

Processes 4 to 8 were implemented by the author using the C++ programming language and the OpenGL graphics library. Future Processes will be implemented using different programming languages and may be implemented by people other than the author. The Process descriptions refer to the Elements listed above.

Process 8 A rectangular surface densely filled with instances of Element 2, each with a different size and speed. Display the intersections by drawing a circle at each point of contact. Set the size of each circle relative to the distance between the centres of the overlapping Elements. Draw the smallest possible circle as black and largest as white, with varying greys between.

Process 7 A rectangular surface filled with varying sizes of Element 1. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to black



Process 4 (Image 1), 28.625" x 28.625" inkjet print, 2005
 Print derived from Process 4. The Process 4 software was modified to create this dense network diagram.

and the longest to white, with varying greys between. Draw the perimeter of each Element as a white line and the centre as a black dot.

Process 6 Position three large circles on a rectangular surface. Set the centre of each circle as the origin for a large group of Element 1. When each Element moves beyond the edge of the circle, move its position back to the origin. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to white and the longest to black, with varying greys between.

Process 5. A rectangular surface filled with varying sizes of Element 1. Draw the perimeter of each Element as a white line and the centre as a black dot. If two small Elements are touching, draw a grey line between their centres.

Process 4. A rectangular surface filled with varying sizes of Element 1. Draw a line from the centres of Elements that are touching. Set the value of the shortest possible line to black and the longest to white, with varying greys between.

The hardware running this software is inconsequential. In time, it will inevitably fail. It was selected to be as robust as possible with current technology, but contemporary electronics are fragile. If an element of the hardware fails, it can be replaced without diminishing the work. Eventually, compatible components will no longer be available because computing technology is continually shifting. When hardware failure inevitably occurs, a new hardware system will need to be acquired and the software should be rewritten for the new hardware to take advantage of the technical advancements since the inception of the process.

Software as Art

Searching for the history of software as an art medium reveals two distinct groups of progenitors that emerged in the 1960s. There is one category of artists who used early digital computers for the production of their work, and another who explored ideas now associated with the synthesis of software and art, but did so without working directly with the technology. The history of using computers to produce visual images is a largely unrecorded one that is rapidly coming to focus through recent publications and exhibitions. As a result of the obscurity of this work, it has had little impact on contemporary artists working in software.³ Alternatively, we can look at the work of artists in the critical groupings of conceptual art and Fluxus. These artists, including Sol LeWitt, Yoko Ono, La Monte Young, Hans Haacke and John Baldessari, have all influenced the way contemporary artists think about software. Software is an immaterial medium, much like thought, and the work of these artists is extremely relevant to the idea of software as art.

In the 1950s the first images created using computers were generated by scientists and engineers who had access to the scarce, expensive and complex technology. Even by 1969, Jasia Reichardt, then a curator at the Institute for Contemporary

Art in London, wrote: 'So far only three artists that I know have actually produced computer graphics, the rest to date having been made by scientists.'⁴ The works created during this time were typically made as collaborations between technicians at research labs and invited artists. Organisations such as Experiments in Art and Technology (EAT) and the Los Angeles County Museum of Art's Art and Technology programme facilitated these collaborations. A Michael Noll, a pioneer of computer images, is quoted in Gene Youngblood's 1970 book *Expanded Cinema* as contradicting this strategy: 'A lot has been made of the desirability of collaborative efforts between artists and technologists. I, however, disagree with many of the assumptions upon which this desirability supposedly is founded. First of all, artists in general find it extremely difficult to verbalize the images and ideas they have in their minds. Hence the communication of the artist's ideas to the technologist is very poor indeed. What I do envision is a new breed of artist ... a man who is extremely competent in both technology and the arts.'⁵

Every artist must decide whether he or she will work collaboratively or directly with software, and this has a major impact on his or her work. Working directly with code leads to a deeper understanding of the conceptual potential of the medium. The number of artists writing their own software has increased significantly in the last 35 years, especially since the introduction of the personal computer. Another increase in software literacy occurred with the rapid adoption of the Internet in the mid-1990s.

However, while the communities of artists writing software continues to grow, there are many remaining cultural and technical barriers to be removed. One of the largest obstructions is the lack of tools for writing software created for the needs of artists. Existing programming languages and environments have been written by software engineers to meet their specific needs, which have always been different from the needs of artists. The open-source software movement has provided a way for artists to collaborate on the production of their own tools. It is my hope these open-source art software initiatives⁶ will serve as a catalyst for a dramatic shift in the use of software within the arts. **Δ**

Notes

1 BASIC is a programming language invented in 1964 to teach programming to nonspecialists. Variations of BASIC were distributed widely with early personal computers. Logo is a programming language often used to teach children concepts of geometry. The Logo turtle is used to draw lines onscreen in response to simple commands.

2 See <http://artport.whitney.org/commissions/softwarestructures/>.

3 CEB Reas, 'Who are the progenitors of the contemporary synthesis of software and art?', *The Anthology of Computer Art*, Sonic Acts (Amsterdam), 2006.

4 Jasia Reichardt, *Cybernetic Serendipity*, Frederick A Praeger Publishers (New York), 1969, p 71.

5 Gene Youngblood, *Expanded Cinema*, EP Dutton & Co (New York), 1970, p 193.

6 See <http://www.artsoftware.org>.