Programming Media

Casey Reas

The design of software is a defining factor in modern culture and is increasingly becoming a basis for our reality. Citizens of the world unite in spending their lives staring into the reflective surfaces of their mobile phones and desktop computers. Their minds and hands operate in the space between reality and the arbitrary rules of menus, windows, clicking, and dragging. Artists utilize software to comment on our increasingly digital social and political structures and to challenge the underlying formal assumptions of computer code. Regardless of the content or intent of their work, contemporary artists are expressing their ideas through the medium of software. With the continually shifting focus of the electronic arts (Cybernetics, Virtual Reality, CAVE, A-Life, Net.art, Augmented Reality), software provides the foundation on which meaning and content are constructed. With the revitalization of the concept of "software art" at festivals such as the Transmediale and READ_ME, a critical discussion is emerging around the role of software within our culture and art practice. This essay extends the discourse and focuses on the concept of software as a medium capable of unique expressions and programming languages as materials with specific properties.

Software Defined

What is Software?

Software is written in programming languages, sequences of alphanumeric characters and symbols composed according to rigid syntactical rules.[1] If you do not normally see computer programs, here are a few program fragments for reference:

Perl

```
opendir(DIR, $dir) || die $!;
@files = readdir(DIR);
closedir(DIR);

foreach $file (@files) {
   if($file =~ ".xml") {
               handle("$dir/$file");
   }
}
```

C++

```
            main() {
   int c;
   c = getchar();
   while(c != EOF) {
   putchar(c);
      c = getchar();
   }
}
```

LISP

```
(define (square x)
   (* x x))
(define (sum-of-squares x y)
   (+ (square x) (square y)))
```

Through writing software, computer programmers describe structures that define "processes." These structures are translated into code that is executed by a machine and the processes are carried out by actively engaging the electronic matter within the computer. Massachusetts Institute of Technology computer scientist Harold Abelson explains, "Processes manipulate 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." It is this active process of reading, manipulating, and storing data, that enables the unique aspects of the software medium.

Software is a Medium

Software has enabled a way to build a bridge between the art of the past and the electronic arts of the present and future. As articulated by Roy Ascott, we have transitioned from "content, object, perspective, and representation" to "context, process, immersion, and negotiation." The most unique aspect of software as a medium is that it enables response. A responsive artifact has the ability to interact with its environment. Artificial reality pioneer Myron Krueger suggests a number of interesting metaphors for interactions between people and software including dialog, amplification, ecosystem, instrument, game, and narrative. I am interested in addressing expressions of software that are more fundamental than those discussed by Ascott and Krueger. These expressions are the foundation of the software media and include dynamic form, gesture, behavior, simulation, self-organization, and adaptation.

Each Language is Unique

Just as there are many different human languages, there are many different programming languages. In the same way that different concepts can be conveyed through diverse human languages, different computer languages allow programmers to write diverse software structures. Just as some expressions are not translatable from one human language to another, programming structures often cannot be translated from one machine language to another. Some programming languages were built specifically for business applications (COBOL), some for artificial intelligence exploration (LISP), and some data manipulation (Perl), and many of the structures written within these diverse languages can only be expressed within that language. The abstract animator and programmer Larry Cuba describes his experience, "Each of my films has been made on a different system using a different programming language. A programming language gives you the power to express some ideas, while limiting your abilities to express others."

Programming Languages are Materials

It can be useful to think of each programming language as a material with unique affordances and constraints. Different languages are appropriate depending on the context. Some languages are easy to use but obscure the potential of the computer and some languages are very complicated, but provide total control through providing complete access to the machine. For example, some programming languages are flexible and others are rigid. Flexible languages like Perl and Lingo are good for quickly creating short programs, but they often become difficult to maintain and understand when programs become large. Programming with rigid languages like 68008 Assembly or C requires extreme care and tedious attention to detail, but the results are efficient and robust. In the same way that the different woods Pine and Oak "feel" and "look" different, software programs written in different languages also have distinct aesthetic gestalts. For example, similar software programs written in Java and Flash have unique differences that are noticed by people familiar with both.

Programming is Exclusive

Many people think that computer programmers are a unique kind of person, different from everyone else. One reason programming remains within the boundaries of this type of personality is that similarly minded people usually create the programming languages. It is possible to create different kinds of programming languages that engage people with visual and spatial minds. Alternative languages expand the programming space to people who think differently. An early alternative language was LOGO, designed in the late 1960s by Seymour Papert as a language concept for children. Through LOGO, children are able to program many different media including a robotic turtle and graphic images on screen. A contemporary example is the MAX programming environment developed at IRCAM by Miller Puckette in the 1980s. MAX has generated enthusiasm from thousands of artists who use it as a base for creating audiovisual software and installations. The same way the graphical user interfaces (GUIs) opened up computing for millions of people, alternative programming environments will continue to enable new generations of artists working with software.

Software Expressions

When computer programs execute, they are dynamic processes rather than static texts on the screen. Core expressions of software including dynamic form, gesture, behavior, simulation, self-organization, and adaptation emerge from these processes. These and other basic expressions are the fundamentals on which more complex ideas and experiences are conveyed. Each expression is discussed below and illustrated with an example from the Aesthetics & Computation Group at the Massachusetts Institute of Technology. These examples were created by hybrid artist/programmers from 1998 – 2001 and provide clear demonstrations of software expressions.

Dynamic Form

Dynamic form is form that changes in time. If this form reacts to stimuli, it is responsive. *Scratch* by Jared Schiffman (Figure 1) demonstrates basic qualities of dynamic form. In this software, the position of a controllable circle continuously affects the contour of each visual element. Scratch augments the visual communication of the form by adding layers of movement and fluid response. In general, form can respond to any signal from the environment including common input devices such as a mouse, microphone, and video camera to more exotic devices such as radiation sensors and sonar.

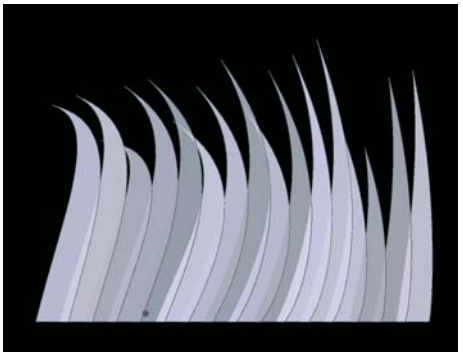

Fig. 1: Jared Schiffman: *Scratch*

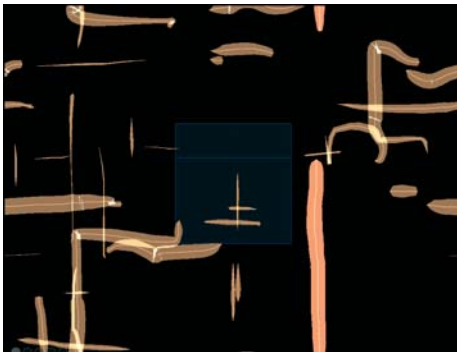

Fig. 2: Golan Levin: *AVES*

Gesture

Gesture is a critical aspect of every continuous medium and software has the ability to convey and interpret gesture. The *AVES* software by Golan Levin (Figure 2) is a group of applications that amplify hand gestures by processing their data as sound and image. One application maps the structure of each gesture into sounds that reflect its degree of curvature. Another layers gestures to create gradual sound textures that activate and combine with the presence of the cursor. Interpreting gestures is more complex, but opens new opportunities for engaging interaction. Handwriting recognition software is one application of gesture interpretation. Some installations and video games use a more basic form of gesture recognition to allow people to direct action with complex motions.

Behavior

Behavior is movement with the appearance of intent. Combining simple behaviors can create personality or disposition. Behavior can be created by intuitively writing programs or by implementing biological models. In the project *Trundle* (Figure 3), the physical object has a program that determines how it should move when presented with stimuli in its environment. *Trundle* searches the environment looking for people, but when it finds someone it attempts to flee. It is curious and timid. An array of sensors on *Trundle's* body continually monitors its immediate environment and sends signals to the micro-controller that determines how the motors should turn. In general, behavior can be used to actively engage the mind through personifying objects, developing characters, communicating affect, and adding a layer of psychological interest to a piece of software.

Simulation

Simulated aspects of the physical world provide an easy access point for perceiving works of software. Our senses have evolved to respond to the rules of the natural world. One of the first computer games, *Pong,* was a highly abstracted simulation of tennis. Modern engineering and scientific communities utilize models of reality as a basis for designing physical objects and conducting research. The *Floccus* software by Golan Levin (Figure 4) creates a group of elegant lines, each a connected list of simulated springs. The undulating movement created by this simple simulation generates awe in spectators when it is combined with response. The lines stretch and contract according to force, mass, and acceleration. In software, simulation can go beyond mimicking perspective, materials, and physical laws—the processes of natural systems can be simulated as well.

Fig. 3: Casey Reas: *Trundle*

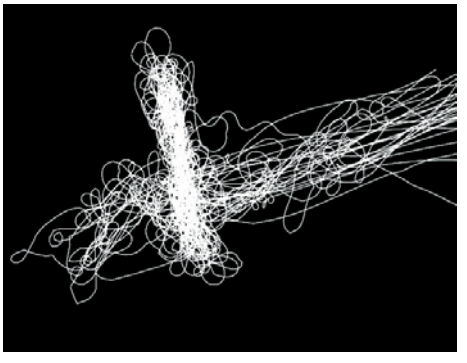

Fig. 4: Golan Levin: *Floccus*

Fig. 5: Ben Fry: *Valence*

Fig. 6: Ben Fry: *Anemone*

Self-Organization

The ability for elements to self-organize makes possible the phenomena of emergence. Structure emerges through the interactions of many autonomous processes. *Valence* by Ben Fry (Figure 5), reads the text of a book word by word and spatially organizes it according to a system of rules. A complex volume emerges from the relations of diverse words in the text. Small changes to the rules of interaction make potentially large changes in the processed visualization.

Adaptation

Adapting is the ability to change. For software to adapt, it must have a representation of itself and be aware of its context. The *Anemone* software by Ben Fry (Figure 6) is able to monitor its density and prune its structure to maintain equilibrium. Anemone is a visualization of website traffic and as the hours and days progress, the software removes sections of its mass to allow for new sections to grow without obscuring legibility of the information. Writing software that truly adapts to its context is a challenge and adaptive expressions are rare. Using an interpreter, it is possible for a program to modify its own program while it is running.

Programming

Although software is consistently utilized within the electronic arts, individuals choose to construct their code in radically different ways from writing in low-level languages to collaborating with programmers. Some artists use software as a tool for creating work in other media. They use commercially available products for generating prints and videos and for making rough sketches that are executed in analog media. Others collaborate with professional programmers by creating specifications that are then implemented by the programmers. Many other people use programming environments developed for designers and artists. They create their work with scripting and visual programming environments such as Director, Flash, and MAX that make it easier for non-programmers to construct software. The smallest group of artists working with software use programming languages developed for professional programmers. They use general languages like C, Java, and Perl and often develop their own custom tools for working within these environments.

There is no correct way to work with software. It is an individual choice balancing control

with simplicity. Mastering programming takes many years of hard work, but understanding the basic principles of the medium is within everyone's grasp. In my opinion, every artist using software should be software literate. What does literacy mean within the context of software? Alan Kay, an innovator in thinking about computation as a medium, has written: The ability to "read" a medium means you can access materials and tools created by others. The ability to "write" in a medium means you can generate materials and tools for others. You must have both to be literate. In print writing, the tools you generate are rhetorical; they demonstrate and convince. In computer writing, the tools you generate are processes; they simulate and decide.
These processes that simulate and decide are the essence of software and they can only be fully understood through constructing them.
Artists are increasing writing their own software. With the growth of the web, the popularity of scripting environments like Flash, and the falling price of hardware, many more artists are exploring programming. The area of audiovisual programming is an excellent example of this trend. Small software companies like Cycling '74, the developer of Jitter, are very responsive to their community of artists and foster the development of enabling tools. Their Jitter tool is a sophisticated library of visual structures for integrating image with sound. Many artists have moved beyond relying on developers for their tools. The Pink Twins, a duo of musician/programmers from Helsinki, have created Framestein, video processing software that links to PD, open source real-time software for performance. The German artist collective Meso has gone even further with vvvv, an ambitious library of tools for real-time video synthesis. Some artists develop software tools for themselves and after a period of refinement, choose to release it to the community.

Synthesis

Over the last thirty years, artists have created innovative work with the aid of the software medium, but they have explored only a small range of the conceptual possibilities. Historically, programming languages and environments encouraged a specific methodology that did not engage the majority of artists who were interested in creating interactive and programmatic work. New tools are emerging that encourage artists to begin working directly with the software medium. The proliferation of software literacy among artists will increase the sophisticated use of software and contribute to new forms of software materials and development environments. These materials and environments have the potential to open the creation of software to an even larger creative and critical community.

1 There are exceptional programming languages called "visual programming languages" that allow structures to be defined with graphic symbols.

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

Ascott, Roy. "Moist Ontology." Published in *The Art of Programming.* Sonics Acts Press, Amsterdam. 2002

Cuba, Larry. "Calculated Movements." Published in *Prix Ars Electronica Edition '87: Meisterwerke der Computerkunst.* Verlag H.S. Sauer. 1987

Kay, Alan. "User Interface: A Personal View" in *The Art of Human-Computer Interface Design,* edited by Brenda Laurel Addison-Wesley Publishing Company, Reading MA. 1989.

Krueger, Myron. "Responsive Environments." Published in *Multimedia, From Wagner to Virtual Reality.* Edited by Randall Packer and Ken Jordan. W.W Norton & Company, Inc., New York. 2001