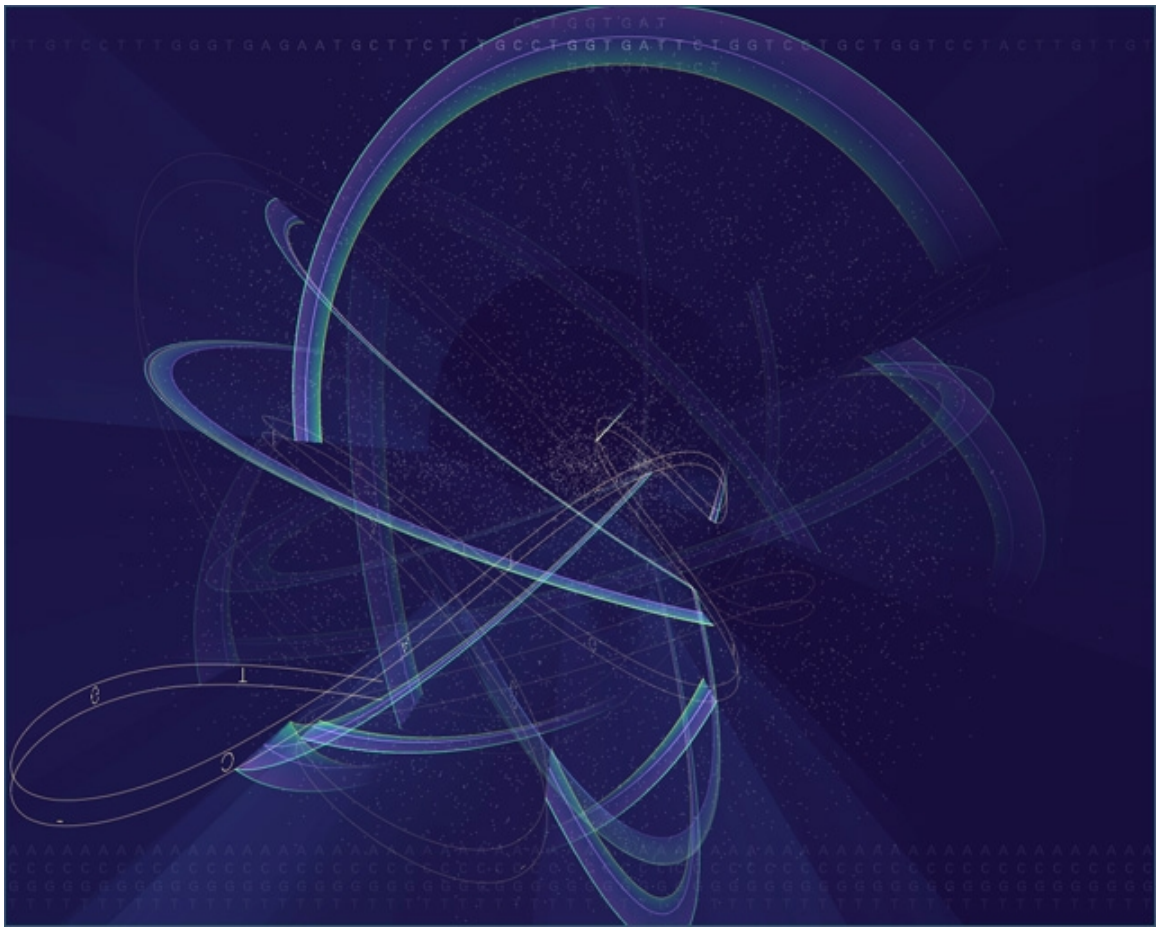
Art Created from a Code

A critical discussion on the fundamental nature of computational art and possible ways to approach, understand and describe it.



Benjamin Fry, Genome Valence. Visualization of the genetic code.

Marie Polakova
BA Digital Screen Arts
University College for Creative Arts
2008

“All things in their fundamental nature are not namable or explicable. They cannot be adequately expressed in any form of language.”
(Ashvaghosha,1st century CE)

Contents

4 Introduction

6 Chapter 1 [new paradigm]

6 1.1

9 1.2

12 Chapter 2[in the beginning was the word]

25 Chapter 3 [processes that simulate and decide]

25 3.1

31 3.2 Few words about computer programming

36 Chapter 4 [art created out of code]

44 Conclusion

45 Bibliography

Introduction

The first chapter of this dissertation “New Paradigm” outlines the new paradigm which emerged with the development of computer technology.

The second chapter “In the Beginning Was the Word” discuss the historical context of software. It is focused on the ancient religious and magical teachings where language is consider to be the tool, the material and the media of Creation. It sketches interesting similarities between the ancient knowledge and discoveries of science. It emphasises the parallels with computer programming.

The third chapter “Processes that simulate and decide” discusses the software art in more detail, and on a perhaps less philosophical basis than the previous two chapters. The second part of this chapter describes the process of computer programming and illustrates it with examples.

The fourth chapter “Art created out of Code” outlines the results of research I have undertaken in order to create a categorization of sub-disciplines of software art. Case studies are used to illustrate the research.

The conclusion provides the summary of the issues discussed in the dissertation.

The discussion of my dissertation is based on and inspired by the works of various authors, listed in the bibliography section. The most influential are perhaps:
Florian Cramer -
A course director of the Media Design M.A. programme at Piet Zwart Institute. He has a background in comparative literature and art history combined with practical experience in computer programming, Unix computing, software and copyleft culture, experimental arts, poetics and aesthetics. In his book *Words Made Flash* and other shorter essays, he discusses the importance of code in software art paradigm, and claims that the computer code is an elemental

component of any digitally produced and reproduced art work. He considers software to be a cultural practice and relates its roots to literature and magical and mystical practices.

John Maeda -

A programmer, designer, artist and professor at MIT Media Lab. His work has been recognized and rewarded by numerous international awards. He is committed to blurring the borders between art and technology, promoting computer programming as a powerful tool for artistic creation. According to him even the programming itself can be redesigned and become accessible for artistically rather then mathematically minded people. In his own words:

> "A major flaw in programming methods is the vast chasm that separates the program's cryptic codes and its graphic output. There is no greater need for visual design than rethinking and redesigning the programming itself."(Maeda 2000 : 406)

He discusses these issues in various theoretical works he has written.

Cassey Reas and Benjamin Fry -

Former students of John Maeda. Currently they are both renowned artists, programmers and educators. During their stay at Media Lab MIT they started to develop 'Processing', programming language aimed at being easily usable for artists and designers, and to teach computer programming within an artistic context. In their book, *Processing,* among the explanation of the actual programming language, they discuss the theme of computer programming within the art and design paradigm.

Fritjof Capra -

Ph.D., physicist and systems theorist, is a founding director of the Center for Ecoliteracy in Berkeley, California and beside his scientific research papers, he has written books about the parallels of Eastern mysticism and modern science and he is an important person within the system theory research.

Chapter 1
[new paradigm]
1.1

> "True digital forms are ephemeral- non existent in the physical realm. To truly appreciate them we must entertain the eye for the invisible-to see into the expansive electrical conscious of the computer. Our ability to comprehend its multidimensional thinking patterns will require intense inquiry into the very nature of computation.
> The common perception of the computer as an object with screen, keyboard, and mouse […] must defer to the computer's rightful identity as pure conceptual mass."
> (Maeda, 2000: IV)

More than fifty years after computer art was born, it is still discussed in analogy (and in contradiction to) non digital art forms. Its terminology is derived from non-digital art practices. Such as terms computer *graphics,* computer *animation,* computer *music,* where only the addition of the word "computer" suggest that something else than traditional-non digital graphics, animation or music is considered. And we would rarely see computer artworks being compared to other computer artworks, let alone computer artwork being discussed as a realm in its own right. Why is it so difficult to grasp and define the paradigm of digitally produced art?

Most commonly, the digital - and therefore software art would be approached from two points of view. The first focuses on the "output" i.e. that which is perceivable by the viewer. The most common way of displaying digital art being the (computer) screen, therefore it is often discussed as part of "[new] media art" and related to video art and even film i.e. to other works commonly displayed on screens. Categorisation would be based on the media and methods of distribution, for example Net Art, computer graphics, computer music and so on, all of which are focused on how and in which medium the work is displayed, rather than on the material and processes which compose them.

As Florian Cramer puts it

> "While software, i.e. algorithmic programming code, is inevitably at work in all art that is digitally produced and reproduced, it has a long history of being overlooked as artistic material and as a factor in the concept and aesthetics of a work."
> (Cramer& Gabriel,2001:1)

In contrast, the second, rather technological approach, focuses mostly on the material – on the code. The "output" is not considered important and in some cases can be ignored altogether and the code itself is considered to be the artwork.

Whilst either of these approaches may be considered appropriate within their own point of view, they are incomplete when software art is considered in a holistic manner. Software art is neither a technology nor a previously known artform. It is a new paradigm, or at least a part of the new paradigm that will be discussed later in this chapter. Cassey Reas and Benjamin Fry describe it so:

> "Software requires its own terminology and discourse and should not be evaluated in relation to prior media such as film, photography, and painting. History shows that technologies such as oil paint, cameras, and film have changed artistic practice and discourse, and while we do not claim that new technologies improve art, we do feel they enable different forms of communication and expression."
> (Reas & Fry, 2007 :1)

The following points define some trains of thought which should, perhaps, be considered when discussing digitally produced and reproduced art.

1. Computational artwork is a continuous, inseparable process.

Within the digital realm, nothing simply just "is". Digitally produced and reproduced artwork is a part of an ongoing process, a constant stream of communication between the various layers of code, electrical impulses and physical mass. S. Snibbe in his essay entitled "The emptiness of code" describes it as follows:

> "It is our consciousness that creates artificial categories from interdependent continuum of existence. Within this framework, computation is understood as an interdependent chain of cause and effect, with no original or primary cause. No part of the continuum from programmer to program, processor to display,

and display to viewer can be removed without breaking the computational chain.[…] Computational artwork exists only as a continuation of the programmer's thoughts through the computational medium and into the mind of the viewer. There is no way to remove or separate these components." (Snibbe Scott in Maeda, 2004 : 228)

The concept of existence as a continuous interdependent process has been accepted by (some parts of) Western science for less than a century but it has been present in religious and mystical teachings for millennia. For instance, Buddhism is based on this concept and its teachings explain the principles with great depth of understanding.

Therefore in this aspect, the full understanding of the computational art requires us to accept these principles, whether the understanding is based on system theory or Buddhism.

2 Software is a cultural practice and a written, literary medium

Digitally produced and reproduced artwork is made out of code. The code is a written, literary medium. "If "literature" can be defined as something that is made up by letters, the program code, software protocols and file formats of computer networks constitute a literature whose underlying alphabet is zeros and ones." (Cramer&Gabriel, 2001:2) Software is not only algorithms or collections of mathematical formulas. In fact computers can not understand mathematics unless the formulae is written in some suitable programming language i.e. translated into the 'code' understandable by the computer; this is discussed in chapter 3.2 .

"But literature is not only what is written, but all cultural practices it involves - such as oral narration and tradition, poetic performance, cultural politics…"(Cramer :122) On this basis, software can also be considered literature – because it is written and it is a part of a larger cultural practice. From, for example, commonly used words, such as 'to google' or 'to browse' where the phrase illustrates a human activity which is born out of the software use; to formation of political - philosophical movements such as Free Software,

where the central thought of the group is based on the software and the use and the distribution of it.

Due to its elementary structure, which will be discussed in Chapter 2, software can be seen to relate to magical, mystical and religious teachings, where the concept of a word –symbol forming matter has been present for millennia.

3 The ways of perception of computational art do not differ from perception of any other art form

Even if in its other aspects computational art relates to its formation in a manner different from other art disciplines, from the "end" [1] user point of view digital art is perceived as a visual, audio or even tactile experience, similarly to any other artwork. In this aspect, the methods used in (any other) art discourse and critique can be applied to software art as well, and there is no reason to perceive digitally produced and reproduced artwork as "unique" due to the process of its creation.

1.2

Looking at the software art and its problematic theoretical discourse can make us aware of wider phenomena. The 20th century engendered what Vilem Flusser called a "change of paradigms". In his 1991 speech delivered in Prague, he further explained:

> "The division of history of the West into antiquity, the Middle Ages, and the modernity is questionable, but nevertheless not arbitrary. In this case, the issue was a change of paradigms, involving changes in living, feeling, and thinking, changes obvious not only to us, at our historical distance, but also to those affected by them." (Flusser, 2002: 85)

As the Middle Ages transformed into modernity the Aristotelean and Biblical heavenly order fell to pieces. The novel order of Rene Descartes's philosophy and Newtonian physics emerged. As Fritjof Capra states:

> " The birth of modern science was preceded and accompanied by a development of philosophical thought which led to an extreme formulation of the spirit/ matter dualism. […] The philosophy of Descartes was not only important for the development of classical physics, but also had a tremendous influence on the general Western way of thinking up to the present day." (Capra, 1975 : 21)

In the 20th Century's shift from modernity to post-modernity, the previous order also fell apart. "… a single universal world in which the same mathematically formulatable laws are valid everywhere has proven to be temporary" (Flusser, 2002: 89). According to Flusser, the post-modern projection of the world looks approximately like this:

> "We are forced to split up the things and processes of the world into three orders of magnitude. In the medium order of magnitude, which is measurable in our measures, that is, in centimeters and seconds, the Newtonian laws are still valid. In the big order of magnitude, that is, the one measurable in light–years, the Einsteinian rules are valid. In the small one, which is measured in micromicrons and nanoseconds, the rules of quantum mechanics are valid. In each of these three worlds, we have to think differently, try to imagine differently, and act differently.
> And yet we cannot separate the three worlds…"(Flusser, 2002 : 89)

Approaches new to Western culture, such as system theory, have emerged, focusing on 'the pattern which connect' as Gregory Bateson calls it, rather than on definitions of separate parts. In system theory; which is applied in various disciplines from neuroscience, sociology and psychology to computer science, the emphasis is shifted from parts to the organisation (system) of the parts. And the interactions of the parts are not understood as static and constant but rather as dynamic processes.

> "The world thus appears as a complicated tissue of events, in which connections of different kinds alternate or overlap or combine and thereby determine the texture of the whole."(Heisenberg , 1963 : 96)

If the pattern of history repeats itself, which we might expect, the new ways of understanding first established in the field of science will be eventually adopted as general understandings of the whole culture, which will obviously form the approaches of art-theoretical discourse as well.

1
When perceiving software as a continuous process there is obviously not a beginning or an end. I'm using the term "end" for the sake of simplicity.

Chapter 2

[in the beginning was the word]

The history of code

1:1 In the beginning was the Word, and the Word was with God, and the Word was God. [. . .]
1:14 And the Word was made flesh, and dwelt among us[…]
(John 1:1,1:14)

Without paying attention to the theological explanation of the above text, we may understand it, simply, as a description of a transformational process whereby a non-material substance ("the word") is made physical ("the flesh"). The non-visual substance therefore becomes visually perceptible.

The description of such a transformation can be found in religious and/or magical systems of many, if not all, cultures. Teachings, such as John's Gospel quoted above would in general have many layers of meaning, and can also be approached on different levels of understanding, varying from meditative contemplation to mathematical analysis and combinations of the above.

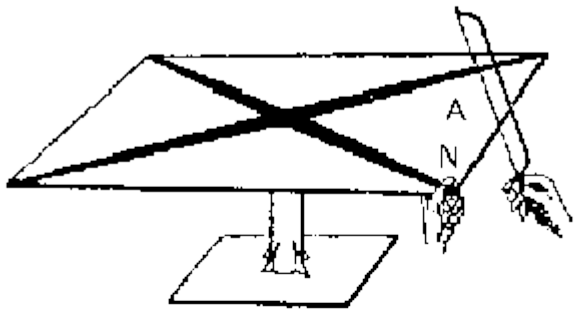
What is "the Word"?

A written word is a sequence of symbols (characters) organized in a particular order, which holds a meaning to those who understand the system of its organization. Without this understanding, the symbols remain graphical elements, having hardly other than aesthetical import. Yet, human language, even in its spoken form, is of a symbolic nature. In order to understand it, one needs to be able to assign the right connotations with the words (i.e. symbols) they just perceive. However, the spoken word is essentially a sound and it has another – a phonic level of meaning, which is comprehensible (as long it is within their hearing range) even for those are not able to "decode" its other layers of meaning. For example, for those who do not understand that particular language.

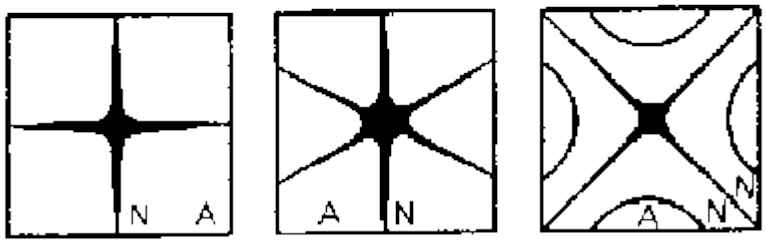
Sound is a vibration, kinetic energy, a waveform. The knowledge that sound can directly affect matter is well explained by science. Already in the 18th Century, the physician and musician Friedrich Chladni observed that when a metal plate covered with sand or other similar substance was made to vibrate by running a violin bow across it, a sound vibration arranged the sand into symmetrical patterns. (Fig 2) This same phenomenon has been researched by Japanese scientist Dr. Masaru Emoto in reacent years. Using high-speed photography he discovered that crystals formed in frozen water have unique forms, which, according to his research, can be affected by exposing the water to sound or even to written words or concentrated thoughts (such as prayer) which might be either vocalized or just meditated. (Fig 3, 4)

These discoveries might appear surprising within still commonly accepted Western dualistic understanding, which places the “matter” and the “spirit”, the “body” and the “mind” into separate, classifiable categories. However, the concept of the thought, the word, the letter or the symbol being interdependently linked with matter, and therefore being able to affect and manipulate it, is at the core of religious, mystical and magical teachings of all cultures. In Budhism this concept is extended even further and the matter is not only considered to be formed by the “word” but to be the manifestation of the “word”.

> “It was taught by the Buddha…the past, the future, physical space,…and individuals are nothing but names, forms of thought, words of common usage, merely superficial realities.” (Madhyamika Karika Vrtti quoted in Murti 1955:198)



(i)



(ii)

Modes of Vibration of Plate - (1790) - Chladni's Figures

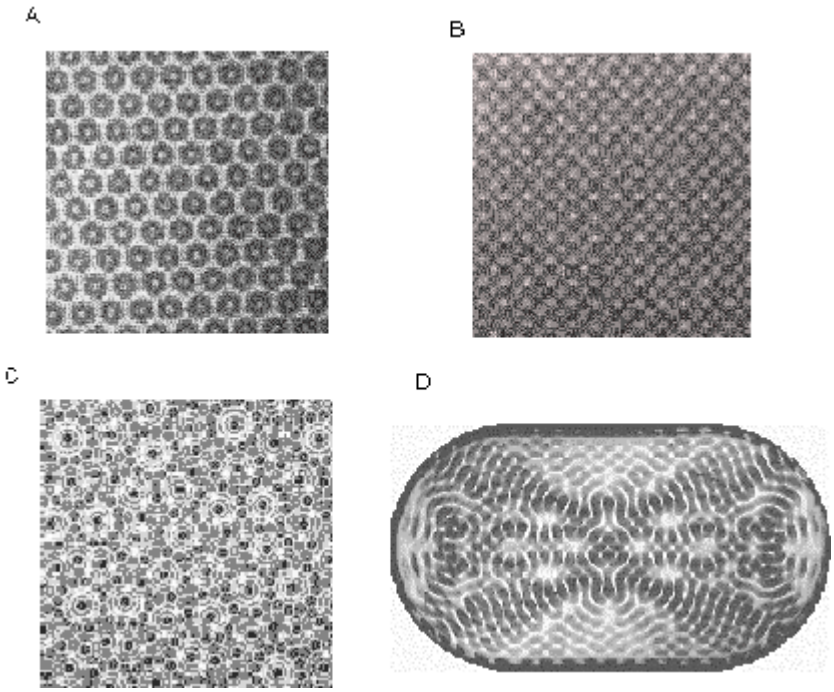


Fig. 2 Chladni figures

Fig.3
Photograph from Dr. Emoto's research. This water crystal was formed after a sample of the water was exposed to the written word ' Courtesy'

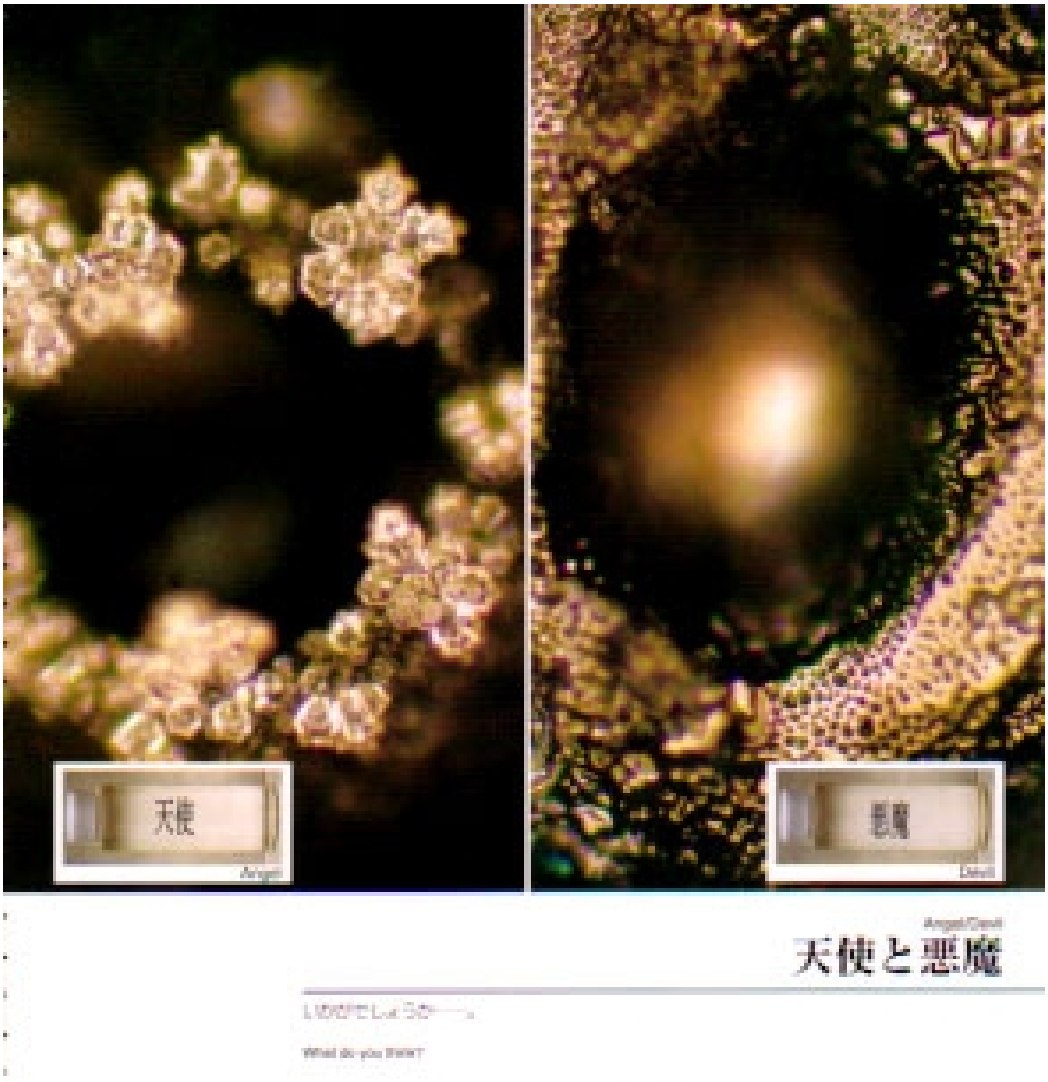


Fig.4
Photograph from Dr. Emoto's research.
Left : water exposed to the word 'Angel'. Right: water exposed to the word 'Devil'

Based on this concept, these teachings understand the process of "shaping the letters so as to form reality" (Blumenthal, 1978:22-29) not in a metaphorical sense, but as a genuine instruction, which when followed correctly will perform a certain task. According to Florian Cramer: "Magic therefore is, at its core, a technology, serving the rational end of achieving an effect, and being judged by its efficiency. " (Cramer, 2005:15)

The magical statements and the computer code have the same purpose. They are tools to manipulate “the reality”. They are not literature in a sense of a poem or a story, with metaphorical and metonymical qualities. The words here serve as signs representing elements (materials and commands alike) of multi-layered, interdependent processes. The execution can be achieved only when certain conditions are met.
In both cases, the cause is the manipulation of “the reality”, from influencing already existing elements to the creation of new, previously non-existant ones. “The technical principle of magic, controlling matter through manipulation of symbols, is the technical principle of computer software as well.” (Cramer, 2005:15). From the moment when the process of execution is set in motion (by turning the computer ON, for example) the words of the programming language are closely bounded with matter.

> “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.” (Reas, 2003)

The words are not anymore dependent on the one who wrote them. “They are real, having existence outside the human mind.” It may be argued that those words are only commands for the computer hardware. But, simply speaking, computer hardware without being programmed is just a piece of material, unable to perform any task. Moreover, computer codes do not mean much without being executed by the hardware which they had to program in order to execute themselves.

To further describe the parallel between computers (and computation) and ancient religious, mystical and magical teachings, the following part of this chapter is focused on Kabbalah - Jewish mysticism. According to Kabbalah the Universe “is built essentially on the prime elements of numbers and letters, because the letters of God’s language reflected in human language are nothing but concentration of His creative energy.”(Scholem, 1971 : 337). The teaching of

Kabbalah exists for over two thousand years and in the same way as other religious teachings, it has been originally passed as an oral tradition. Later, probably in 8th – 10th century it started to be recorded in written form.
Perhaps the most important text of the Kabbalah teaching is the Sefer Yetzirah – The Book of Formation (spelling and translation sometimes vary, other most often encountered is Sefer Yesira translated as "The Book of Creation") The origin of the text is a topic of heated disputes among historians and so far it remains unknown. The text of Sefer Yesirah essentially describes the formal instructions – an algorithm for the creation of the world through letters.

> "According to *Sefer Yesira,* God's "speech" was not talking in the sense of someone speaking, but rather a manipulation of the letters of the Hebrew alphabet. These letters, *Sefer Yesira* teaches, are not merely linguistic symbols. They are real, having existence outside the human mind. They are made of a special spiritual substance and, hence, could be formed, weighed, shaped, etc. by God. Creation, then, was the process of shaping the letters so as to form reality." (Blumenthal, 1978 : 22-29)

The example of Sefer Yetzirah was chosen here, because the descriptions it gives are formal algorithm - the same as the computer programming code. One paragraph, for example, states:

> "From two letters, or forms He composed two dwellings; from three, six; from four, twenty-four; from five, one hundred and twenty; from six, seven hundred and twenty; from seven, five thousand and forty; and from thence their numbers increase in a manner beyond counting; and are incomprehensible." (Sefer Yetzira, IV-4)

What is being described here is nothing else than the mathematical law of permutation, which essentially explains how many possible combinations certain amount of elements can have.
"From two letters, or forms He composed two dwellings" can be written as mathematical formulae: 2! = 2 *1 = 2 and means that two elements allow two permutations,{1,2}and {2,1} namely. "from three, six" or 3!= 3*2*1 = 6 than signify that three elements can have already six permutations ({1,2,3},{1,3,2}, {2,1,3}, {2,3,1},{3,1,2}and {3,2,1}) and so on exactly as described by ancient text of Sefer

Yetzirah. The very same law is used widely in computation, just as the other algorithms described within the book.

What is discussed here, among other issues, is a concept of language being within itself its own instructions and 'material'; the medium and the message alike. Execution of its instructions by itself can transform the language into forms which are not anymore lingual. These concepts would be well known to anyone familiar with computing. For example, one of the essential discoveries of computer science is, that the computer can treat its own instructions as a form of data, which allowed the concept known today as computer programming. As described by Cramer and Gabriel:

> "By running code on itself, this code gets constantly transformed into higher-level, human-readable alphabets of alphanumeric letters, graphic pixels and other signifiers. These signifiers flow forth and back from one aggregation and format to another. Computer programs are written in a highly elaborate syntax of multiple, mutually interdependent layers of code. This writing does not only rely on computer systems as transport media, but actively manipulates them when it is machine instructions. "
> (Cramer & Gabriel,2001:2)

Within Kabbalah, another very contemporary idea is that of a semi-autonomous artificially created but in a sense living being - the 'Golem'.

> " The word 'golem' appears only once in the Bible (Ps. 139: 16), and from it originated the Talmudic usage of the term – something unformed and imperfect. In philosophic usage it is matter without form. Adam is called "golem", meaning body without soul…" (Encyclopedia Judaica vol.7 : 736).

Even though the word was mentioned in earlier sources, it is Sefer Yetzirah which provides instructions (although encrypted) on how to create such a being. There are different explanations of these instructions and various legends of Rabbis who succeeded in creating the Golem - the legend of Rabi Loew's Golem of Prague being probably the most well- known one today. (Fig 5) Essentially the Golem has a physical body, in most legends created out of clay. This remains nothing but a piece of matter until it had been passed the "Shem ha-Meforash , the fully interpreted and expressed and differentiated name of God" (Scholem

1971 : 339)- the code which makes unliving matter alive. Different versions of the legend vary in descriptions of the details - sometimes the letters are inscribed on the piece of parchment, which is placed in Golem's mouth or they are written on his forehead. The 'code' would usually consist of the letters *aleph*, *mem*, *tav*, which is *emet* and means 'truth', when the *aleph* is erased you are left with *mem* and *tav*, which is *met*, meaning 'death' and will render the Golem back to its non-living state. (Fig. 6,7)

Whether the ancient Rabbis did or did not manage to create the 'functional' Golem by following the Sefer Yetzirah instructions can hardly be proved. But when being perceived as a metaphor, we might observe striking parallels between the non-living clay body of the Golem and computer hardware, and between the 'emet' – 'met' letter permutations which sets the matter 'alive' and the computer code. For example, Professor Gershom Scholem, Jewish philosopher and historian coined the name "Golem of our Times" as a term for the computer. The first computer constructed in Weizmann institute (Rehovot, Israel) was according to his wish named Golem1. In his speech "The Golem of Prague and the Golem of Rehovot" delivered at the Weizmann institute, on June 17, 1965, Professor Scholem describes six points to illustrate the parallels between the two "Golems". Two of them being particularly interesting within the context of this dissertation:

> "1. Have they a basic conception in common? I should say, yes. The old Golem was based on a mystical combination of the twenty-two letters of the Hebrew alphabet, which are the elements and the building stones of the world. The new Golem is based on a simpler, and at the same time more intricate, system. Instead of twenty-two elements, it knows only of two, the two numbers 0 and 1, constituting the binary system of representation. Everything can be translated, or transposed into these two basic signs, and what cannot be so expressed cannot be fed as information into the Golem. I dare say the old Kabbalists would have been glad to learn of this simplification of their own system.[…]
> 2. What makes the Golem work? In both cases it is energy. In the old Golem it was the energy of speech, in the new one it is electronic energy."(Scholem, 1971 : 339)

The ancient legend seems to be more alive than we might think at first glance.

Fig 5. Mikolas Ales, 1897. Rabi Loew's Golem.

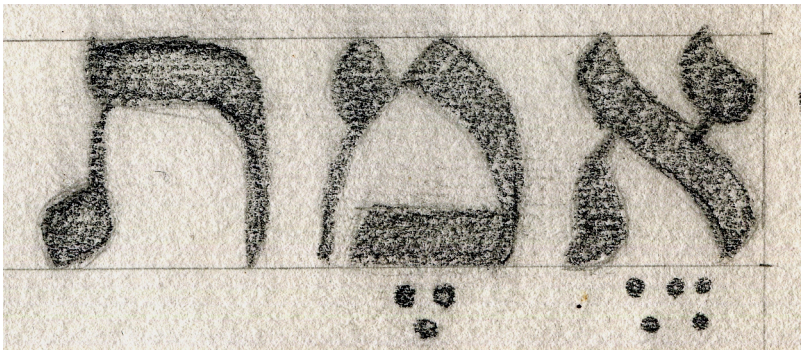


Fig 6. ‘emet’ – truth

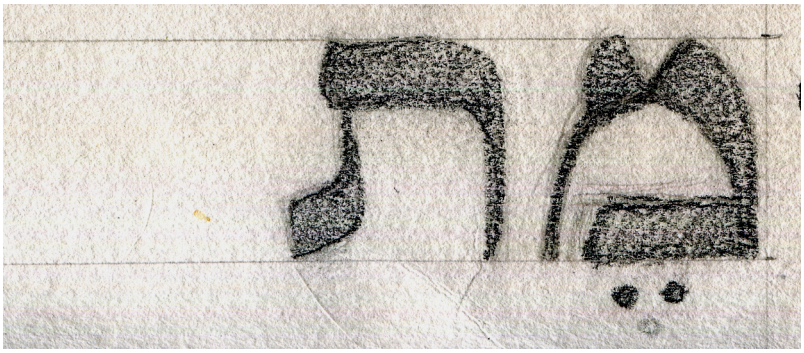


Fig 7. ‘met’ – death

The concept of the language as a tool and material of algorithmic manipulations carries on through Western history. The Kabbalistic principles were adopted by Christian scholars, and scientists, such as Ramon Lull who’s work *Ars Magna Sive Combinatoria (* Fig 8) is considered to be an ancient precursor for Artificial Intelligence, (according to Z. Neubauer). Or 17th century Czech teacher, scientist and philosopher Jan Amos Comenius who used Kabbalistic computational methods to structure his popular encyclopedia “Orbis Pictus”(Fig 9), which was the first illustrated children’s school book in history and, according to Cramer, the graphic user interface of computers used today is based on Comenius’ work. Unlike my previous examples, in Lull and Commenius’ approaches the letter combinations were understood “…as method of logical reasoning, generation and classification of statements and knowledge.” (Cramer, 2005:41)The computational language manipulations were also used in the classical rhetoric and poetics among others.

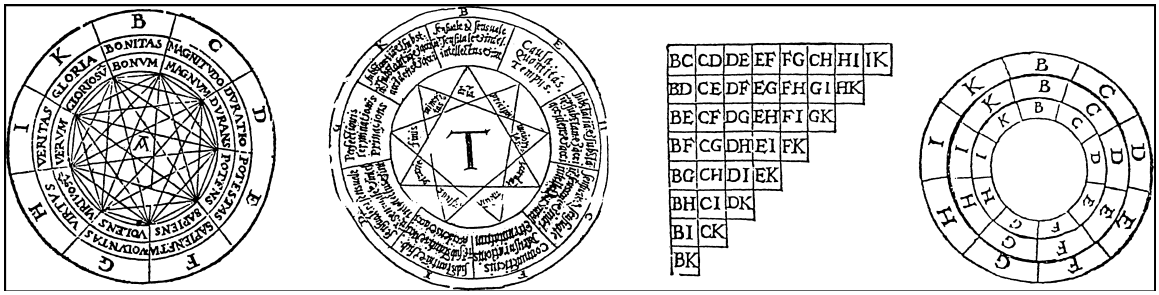


Fig. 8
Ramon Lull's Ars

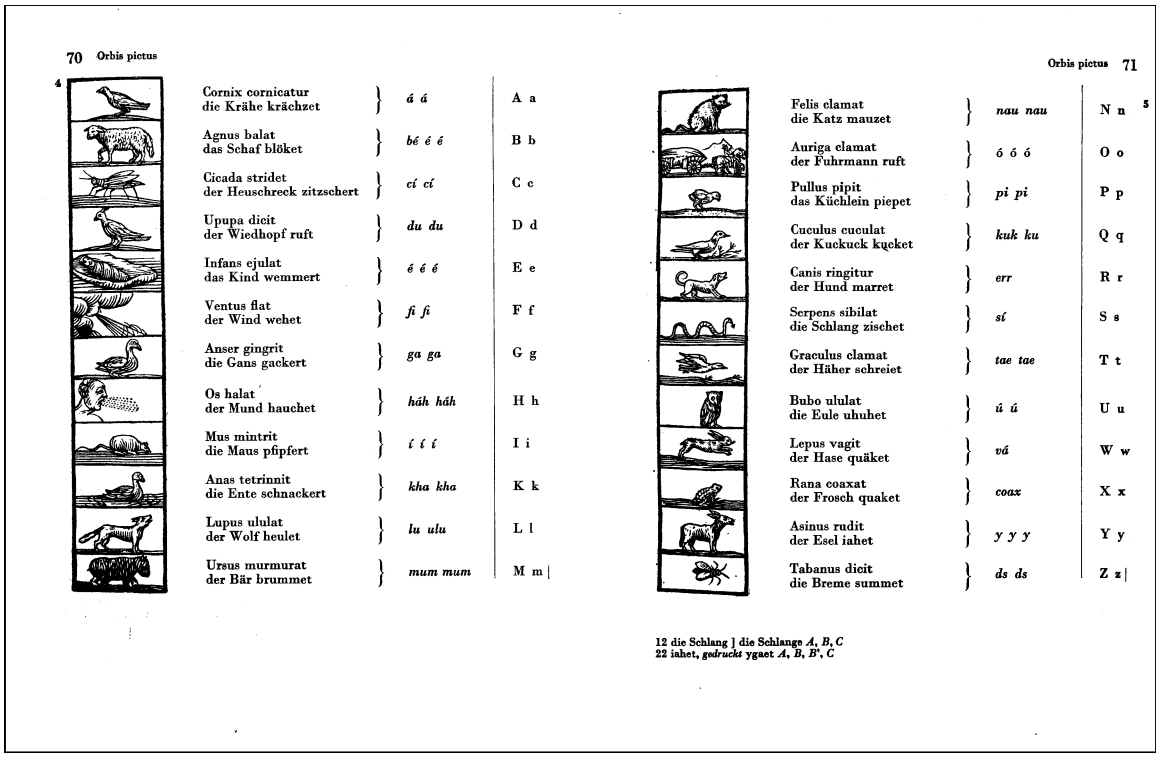


Fig.9
Comenius' encyclopaedia *Orbis Pictus*

Observing the same thought pattern, that of the word (letter, code, symbol) - matter manipulation, being embedded within possibly all cultures and fields of human knowledge, we might ask if the structure of knowledge is not simply reflecting this basic principle, which is woven into the elementary structure of the Universe, perhaps, being the essential principle of Nature and therefore of human beings. What else is the genetic code than a set of instructions (an algorithm) of how to form living bodies from non-living matter?

Chapter 3

[processes that simulate and decide]

3.1

In the 1960s when computer art was born, artists were closely cooperating with scientists (and programmers). They could hardly access computers otherwise, since the large mainframe computers of that time cost anything from $ 100 000 to several million US dollars and only scientists had access to industrial research laboratories and university computer centres, which owned such equipment. (Fig .10) There was obviously no “ready made” software and the only way to realise an artwork within the computer realm was to program it by your self, which at that time was an even more tedious task than it is today, or to collaborate with someone who would program it for you.

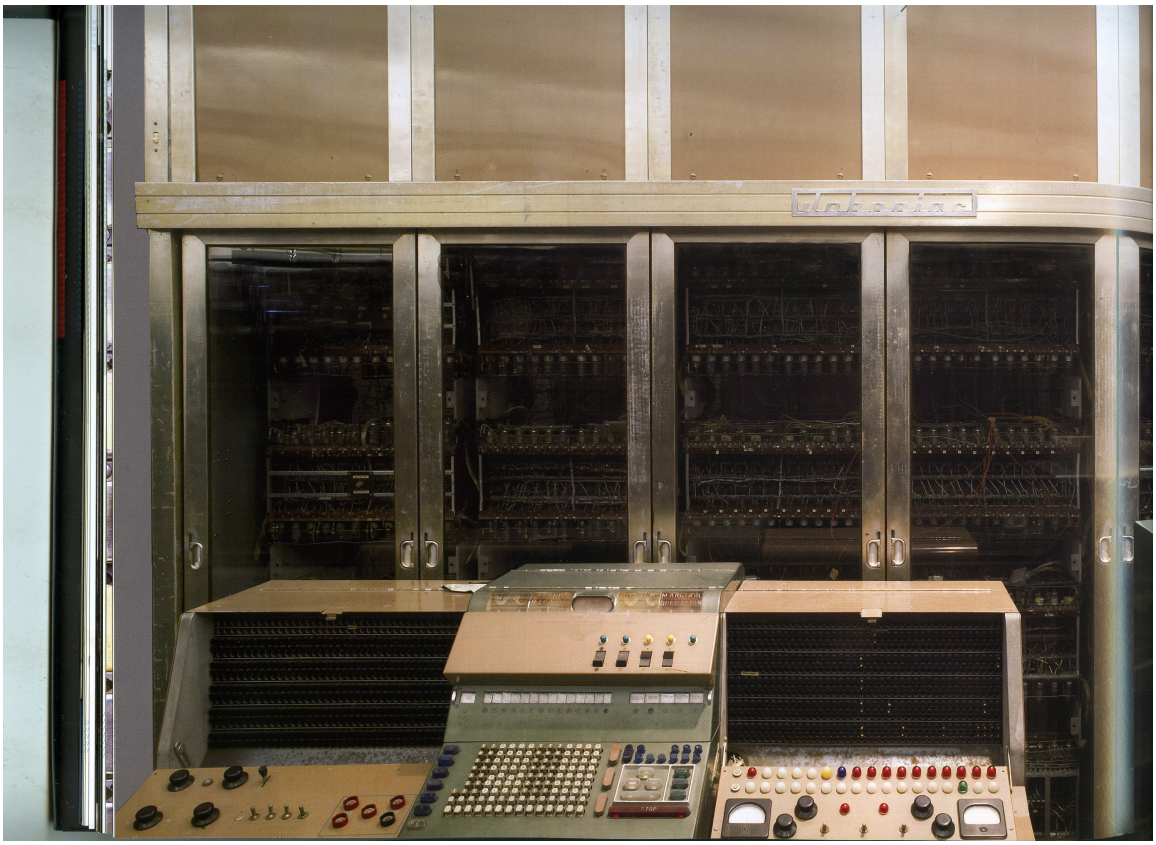


Fig.10
This image depicts part of ‘Johniac’, a computer constructed in 1954 by Rand Corporation. It was valued at $470.000

With the invention of the microprocessor in the mid-seventies, computers gradually became smaller, cheaper and more available. The user interface, in other words, that which is between the computer and the user, has also changed. Starting with the Xerox 8010 Star Information System in 1981, the Graphical User Interface (GUI) became wide spread. While the other user interfaces, the Text User Interface (TUI) and Command Line Interface (CLI) in particular, required excessive knowledge of the operating system, as the commands have to be typed, and therefore known, by the user. GUI allow users with minimal formal training and knowledge to operate the computer. The commands for the computer are 'hidden' behind graphical elements and can be manipulated by simple mouse clicks. The design of GUIs has been inspired by known 'non digital' environments, such as office desktops.

> "To most of us, paper is more of a state of mind than any object-it is a place outside our minds to think and reflect. It is unfortunate that the display technology of the computer we use has been designed around the flat, rectangular metaphor of machine-cut paper, instead of the unflat, unrectangular, and infinitely multidimensional space of pure computation."
> (Maeda, 2000:145)

Even though the GUI is undoubtedly one of the essential factors causing the wide spread of computers, it has also a drawback. The true nature of computation remains hidden behind the desktop, paper and pen metaphors.

> "Word processors are based on typewriters and graphics programs mimic paper, pencils and brushes. However, what program is inspired by a flowing stream?"
> (John Simon Jr. in Maeda 2004:46)

The design of the software environment is nearly always based on the tools and environments from the 'real' world. Word processors mimic typewriters. Graphics software reminds us of the drawing desk of the pre-computer era, with the use of familiar tools such as brush and pencil, and video editing programs are based on the film editing suite. There is a reason for this, as for example, graphics software is intended to help its user to produce images which are perceived by the viewer as graphics, whether created digitally or not. But such a design of the user

interface tends to conceal, and even make inaccessible the ‘material’ - the essentially different nature of digitally produced artwork. Following our example of graphic software, its design doesn’t guide the artist to realize that by using the tool called ‘ink’ he is not, at all, doing anything as drawing with ink on paper (or canvas, as the working surface of graphic software is often called). The seemingly two dimensional ink mark is in fact a manipulation of multi-dimensional computational space.

Behind the visible brush and ink are layers of text- the computer code which, very simply speaking, in its ‘elementary’ machine code phase manipulates the electric circuits of the computer , which then, among other tasks, direct the beam of electrons hitting the chemical surface, covering the glass plate of the monitor, and so to form the visible ‘pixels’- that has the visual reference to the ink known from the ‘real’ world. What a fascinating realm lies behind the blob of digital ink! The computer software ‘Time Paint’ created by John Maeda emhasises rather than hides this nature of digital ‘painting’.(Fig.11)

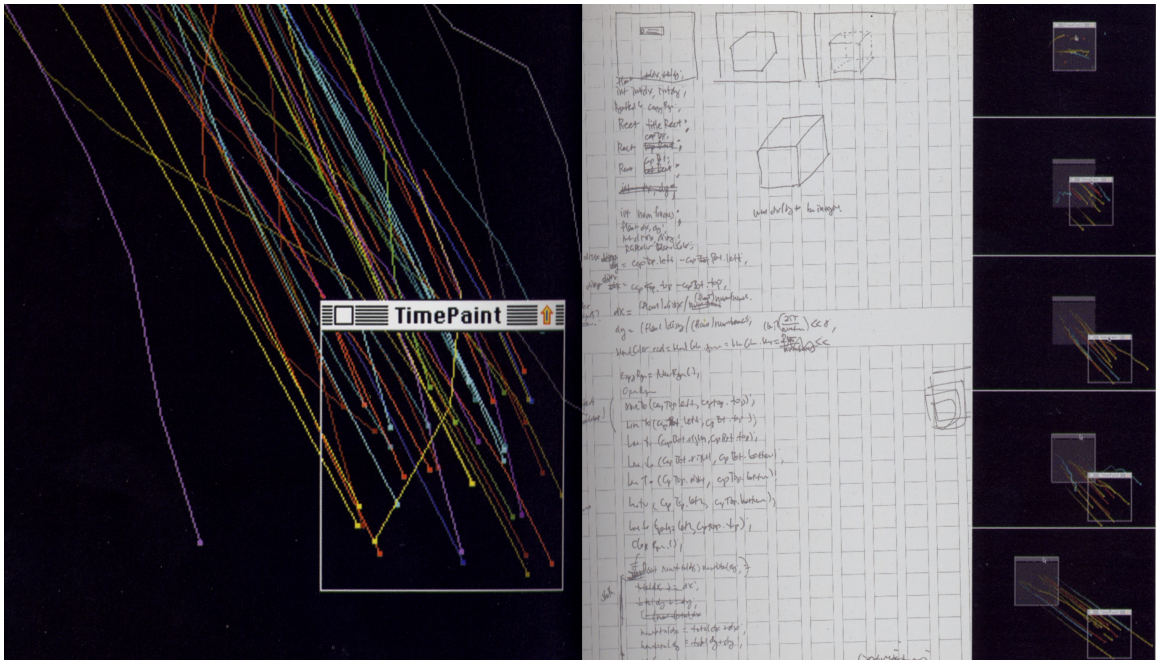


Fig. 11
'Time Paint'
"We usually think of a stroke of digital ink as nothing more than a mark left in space. When its true identity is revealed as a path not only through space but also through time, its sculptural, space-time qualities can be revealed, as visualized in this project…"(Maeda, 2000:99)

Preprogrammed software is shaping the way we use it. It shapes what we can and cannot do with it. The limitations are given by the technology - by the programmer who created the software; while in programming the limitations are within your mind. It's mostly your understanding which limits and permits your action. This is not to say that they are not any limitations within the programming languages. Each of them is the most suitable for different purpose and has its unique qualities. "A programming language gives you the power to express some ideas, while limiting your abilities to express others." (Cuba, cited in Reas & Fry :1)

The tools and materials used for the realization of the artwork are important. Through them the matter of an artist's thoughts is shaped! It is no different within the computational art paradigm. The decision of which programming language to use has the same importance for the artist-programmer, as the decision to use oil or aquarell has for the painter. It can be said that each programming language

produces its unique kind of aesthetic, sometime distinctive enough to be recognizable. Similarly, the distinctive look of oil painting can be hardly confused with charcoal drawing. Also, artists working with code would have their unique personal style, like any other field of art. (see examples provided in chapter 4) Again, the knowledge, the understanding of the artist-programmer is the limitation which permits or stops him to use the particular language. In some cases artists-programmers would even develop their own programming languages to fulfill their needs. For example, graphical programming environments such as Pure Data or MaxMSP-Jitter (Fig 12), have been developed for artistic usage. Within these environments, the user is programming via manipulating graphical elements, and these programming languages are therefore more accessible for those unfamiliar with 'code writing'. Another example is Processing (Fig.13), textual programming language and environment developed by Benjamin Fry and Cassey Reas as an efficient and accessible language for artists and designers and as a tool to teach the fundamentals of computer programming within the artistic context.

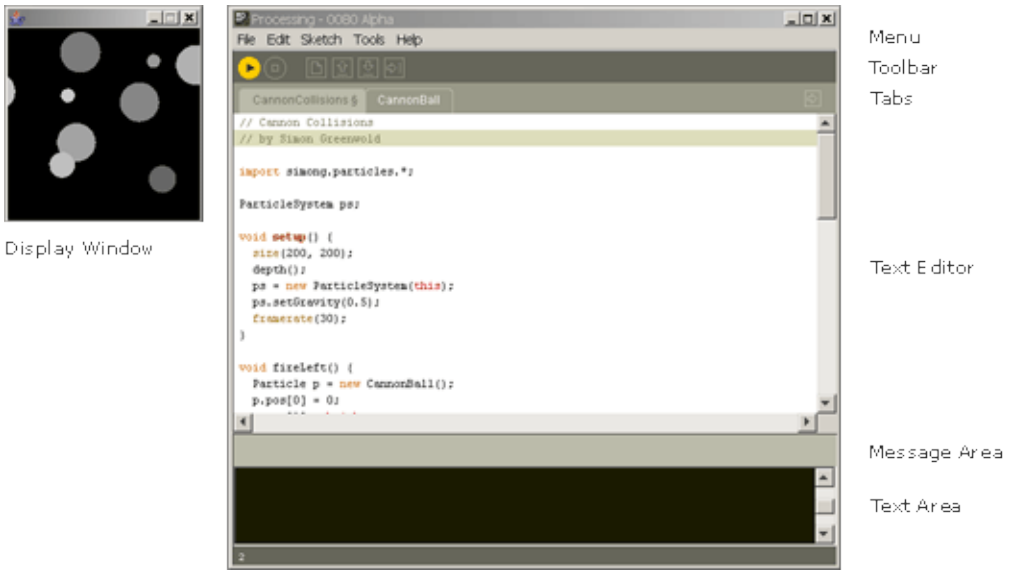


Fig.13
Processing programming environment.

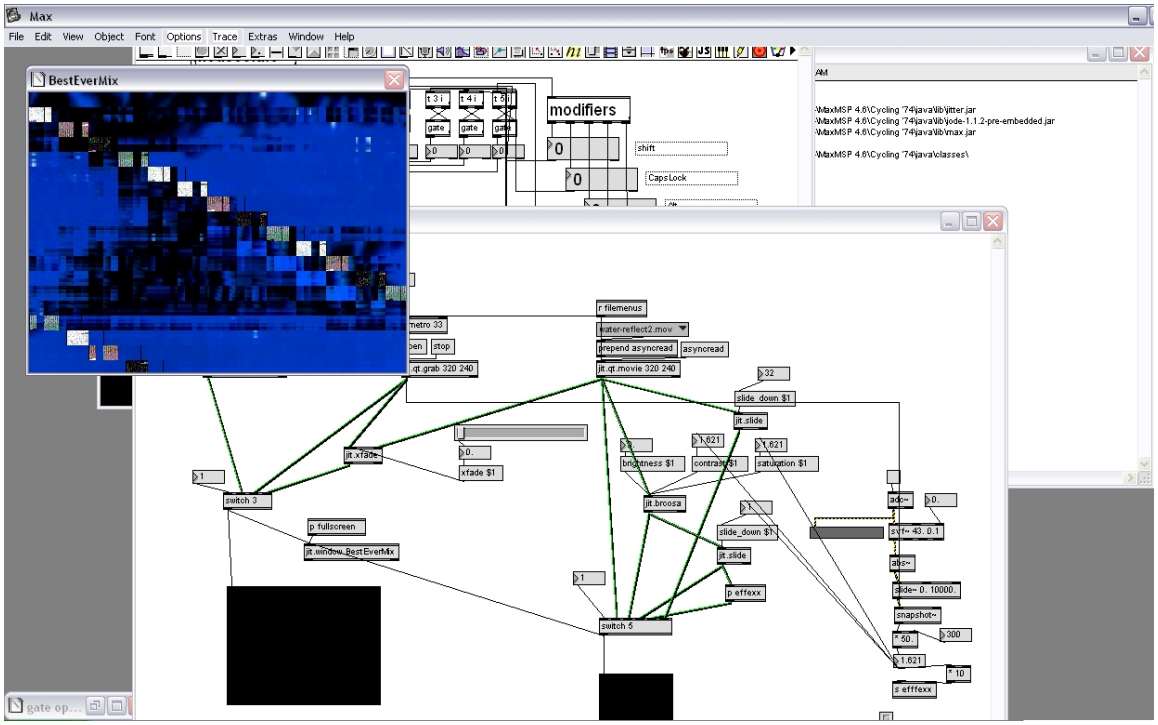


Fig.12
Programming environment of Max/MSP Jitter. Programs call ‘patches’ and each little box signifies an element of the program. Connection between these elements is provided by ‘patchcords’.

The limits given by the technology lay far behind the limits set by the design of pre-programmed software. Therefore creation with the code, from writing their own programs to developing unique programming languages, allow the artists-programmers working within the digital environment to gain freedom which they can hardly have otherwise. In the words of Jered Tarbell, artist and programmer:

> “With software, anything that can be imagined can be built. Software has a mysterious,undefined border. Programming is truly a process of creating something from nothing. “
> (Tarbell, interviewed in Reas & Fry:158)

3.2

Few words about computer programming

> "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."
> Making processes that simulate and decide requires programming.
> (Kay, cited in Reas & Fry :3)

Computer programming is a process of interpreting human thoughts and emotions into instructions that are executable by the computer. When we say or write something, most of the words would trigger visual, or other sensory, imagination in the mind of the person who receives the message, but we would never be in control of what this imagination looks like, we would never actually know. 'Communication' with programming languages follows a similar structure, but the 'imagination' happened in the computer where, unlike in another person's mind, it is accessible to others. John Maeda describes the process of computer programming as "… to unerringly describe the structure of a machine as a sequence of textual codes, which when brought to life in the mind of the computer performs a specific processing task." (Maeda 2000 : 406)

Due to the amazingly complex tasks computers can perform, we might be tempted to think of them as intelligent, nearly human machines. (Un?)fortunately they are not. Underpinning the computer are the electric circuits, which in their simplest form are either "on" or "off". Imagine that you are trying to explain to someone the way to the next town. Imagine doing it, being permitted to use only two words "yes" and "no". In some sense, that is, what computer programming is all about.

Computers can really only "understand" its machine language, code-zeros and ones, which signify "off" and "on" of its electric circuits, but these days probably nobody is writing programs in the machine codes. Usually, the instructions will be written in some of the higher level programming languages. Those are then, by

another program, translated “down” to the machine language. Details of this process vary, depending on the type of the programming language used.
The process of programming would usually start with an understanding of the problem – i.e. understanding the task which the particular program needs to perform, and defining what the solution must do. Later the programmer would think of an algorithm – the general solution for the problem, try to analyze it and verify if the solution really solves the problem. After that the algorithm would be translated into a programming language, executed and tested. (according to Dale Nell & Weems Chip, 2005; 3) Fig 14. illustrates the process of programming, and it is, as artist - programmer and MIT professor John Maeda describes, an “Example of the evolution of an image representing infinity, from a sketch on paper, to an equally rough mathematical model, to an approximated programmatic translation, and finally directly into the computer as a tunable form.”(Maeda 2000: 29)

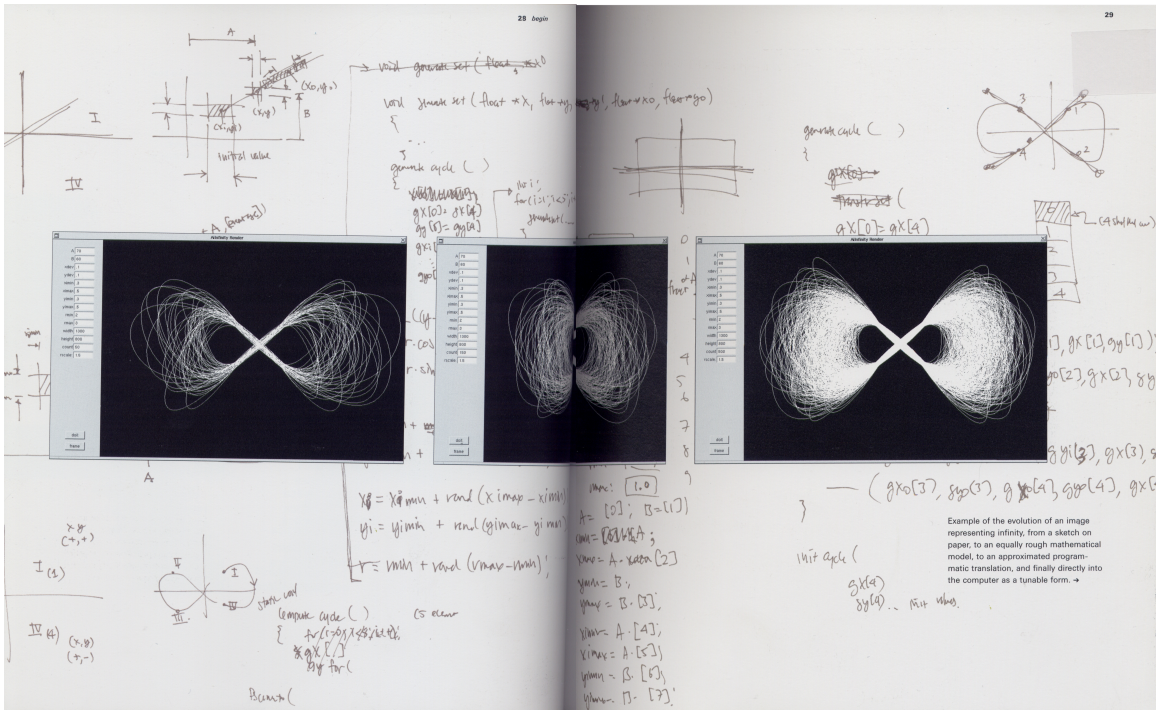


Fig.14
John Maeda. Image representing infinity.

To offer the reader unfamiliar with computer programming better insight, the following paragraph explains the process of computer programming with an even simpler example.

Let's say that we would like to draw an image representing a dark blue square with a thin white line stretching from its top left to the bottom right corner.

As a next step we might think of the process of how this image is drawn, so we can instruct others. We translate the instructions into, so we might ask someone who doesn't know English to follow them. We also want this image to be drawn by the computer and we have chosen to write the instructions in Processing, the programming language discussed in Part 3.1, which is very easy to use for graphical output.

We would specify the first step in English as:

1. Draw a square of a size 5 x 5 cm.

In Czech as:

1. Nakreslete čtverec o velikosti 5 x 5 cm.

In Processing as:

```
//1.
size(200,200);
```

The second step would be in English:

2. Fill it with dark blue colour.

In Czech:

2. Vybarvěte ho tmavě modrou barvou.

In Processing:

```
//2.
background(#0AABFF);
```

And the third, final step is in English:

3. Draw a white, approximately 1mm thin line from left top corner to the right bottom corner.

In Czech as:

3. Nakreslete bílou čáru, přibližně 1mm silnou, z levého horního do pravého dolního rohu.

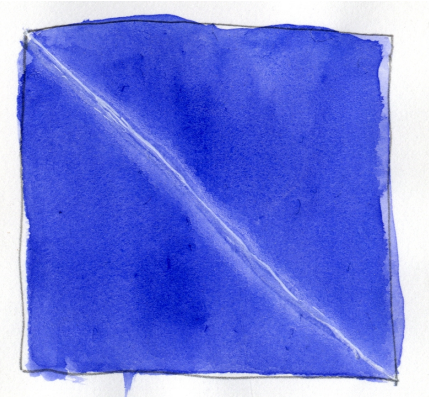
In Processing as:

```
        //3.
stroke(0);
```

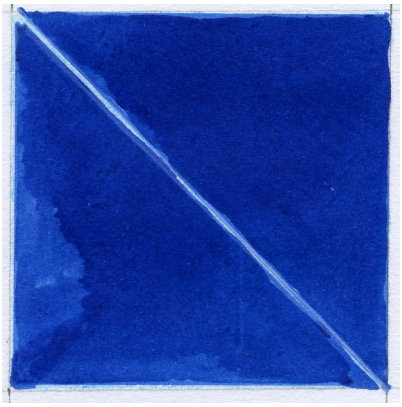
```
strokeWeight(2);
line(0,0,200,200);
```

The following images show the result of following the instructions by two different people and one computer.

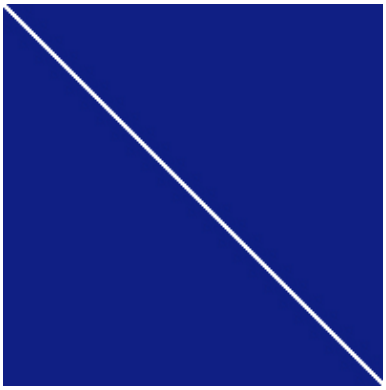
Following the instructions in English result in:



Following the Czech instructions led to this image:



And the computer program resulted in this image:



Principally the instructions can be passed, received and executed only if both the writer and the reader understand the particular language. Otherwise they remain only sequences of meaningless characters. Just as a person who doesn't know Czech wouldn't be able to draw the image following the Czech instructions, the computer cannot 'understand' English or Czech, but it can 'draw' the image following the instructions written in Processing.
Computer programming is often associated with mathematics. Although mathematics is an essential part of the programming code, a computer does not 'understand' it! Mathematical formulae have to be translated into some programming language in order to be understandable by the computer.

This is not to say that computer programming is entirely analogous to the communication in human languages. Programming languages have far more restricted vocabulary and more rigid syntactic and semantic order than any natural language, programming still remains 'unnatural' and difficult to comprehend for many individuals. It confronts the artist with other challenges than the use of other artistic tools and materials. It is more similar to learning a language (even a non- natural one) than mastering the use of tools. However "the foremost challenge in operating such a powerful tool is the same as with the simplest tool: there must always be a clear initial concept that can guide the process to a relevant outcome." (Maeda 2000 : 32)

Chapter 4

[art created from a code]

This chapter is providing an overview of different types of computational art works.

There is not yet any established classification of the types of software art. To create such a system of classification is complicated and my efforts to do so are, of course, tentative. Therefore I want to simply describe different types of software art, without creating overly defined categories, provided with case studies to illustrate the described artforms.

[formal code art]

The code, not the output of the code is the artwork.

In this aspect formal code art relates closely to other written artforms, such as poetry, and to the conceptual art where the notation can not be distinguished from the artwork. Florian Cramer when discussing the *Composition 1961 No.I, January I* created by contemporary composer and former Fluxus artist La Monte Young: “Draw a straight line and follow it.” (Young in Cramer &Gabriel, 2001:2) states: “This piece can be called a seminal piece of software art because its instruction is formal.”(Cramer &Gabriel, 2001:2).

The following example, perhaps more related to prose than poetry is Joshua Samberg’s ‘Self SDK’. The term SDK stands for Self Development Kit and describes, simply speaking, documentation and samples which software engineers need in order to write, build, test, and deploy applications for certain software packages.

Joshua Samberg used C++, a common programming language, to create a functional program, which in itself is a new kind of self-portrait. In the code, he intentionally uses semantically meaningful class, variable, and method names and extensive in-line comments so as to be at least marginally understandable to anyone who understands English, even if they do not have the technical knowledge to fully understand the code.(according to Samberg, 2006)

Similarly to other works of art, code art, such as Self SDK, can also reflect very personal feelings of the artist. In Joshua Samberg's own words:
"Self SDK is an attempt to gain understanding and control of my often difficult and confusing existence by modeling myself and my life in a computer program. As an experienced programmer, I feel a lot of power, efficacy, and safety in the world that lies inside the computer. In real life, on the other hand, I feel hopeless and overwhelmed in the face of the most basic, everyday tasks and occurrences. By constructing a model of myself inside the computer, I have tried to utilize my comfort, experience, and skill with computer programming to explore, understand, and ultimately transform myself." (Samberg, 2006)
(Fig 15,16)

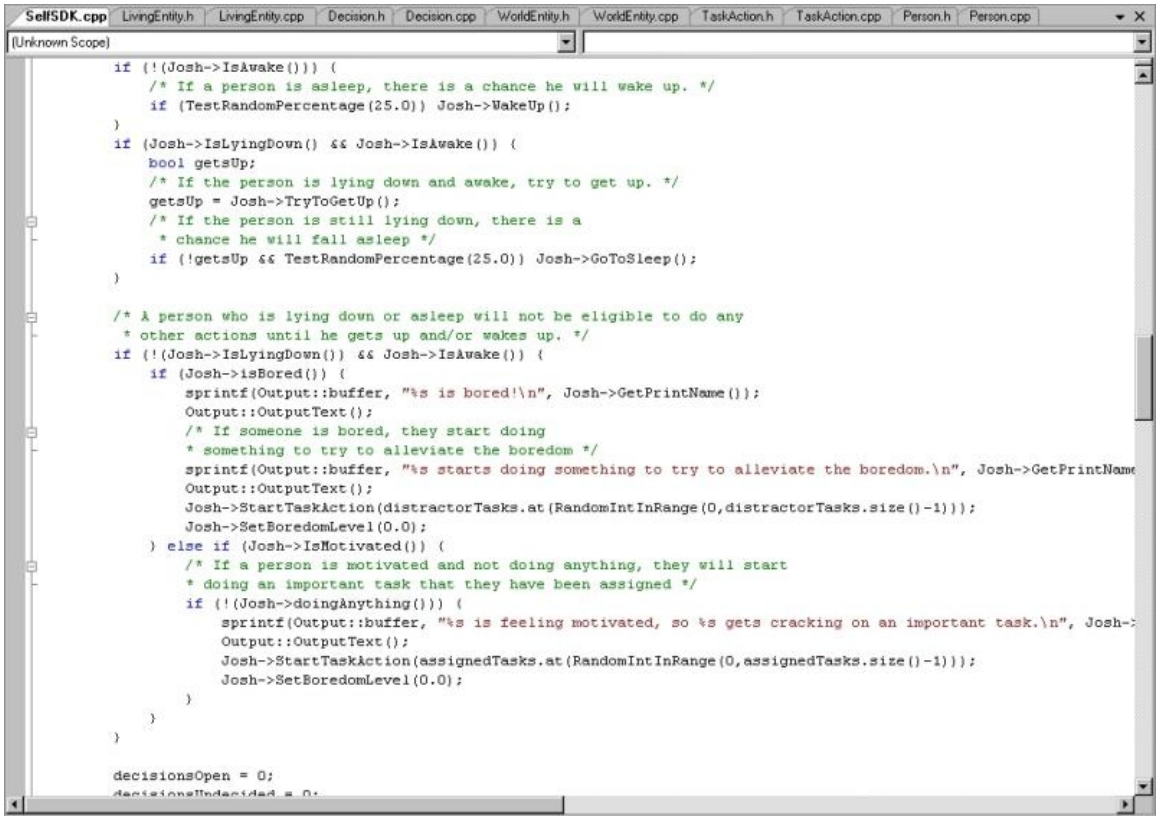


Fig.15
Joshua Samberg, Self SDK program written in C++ programming language.

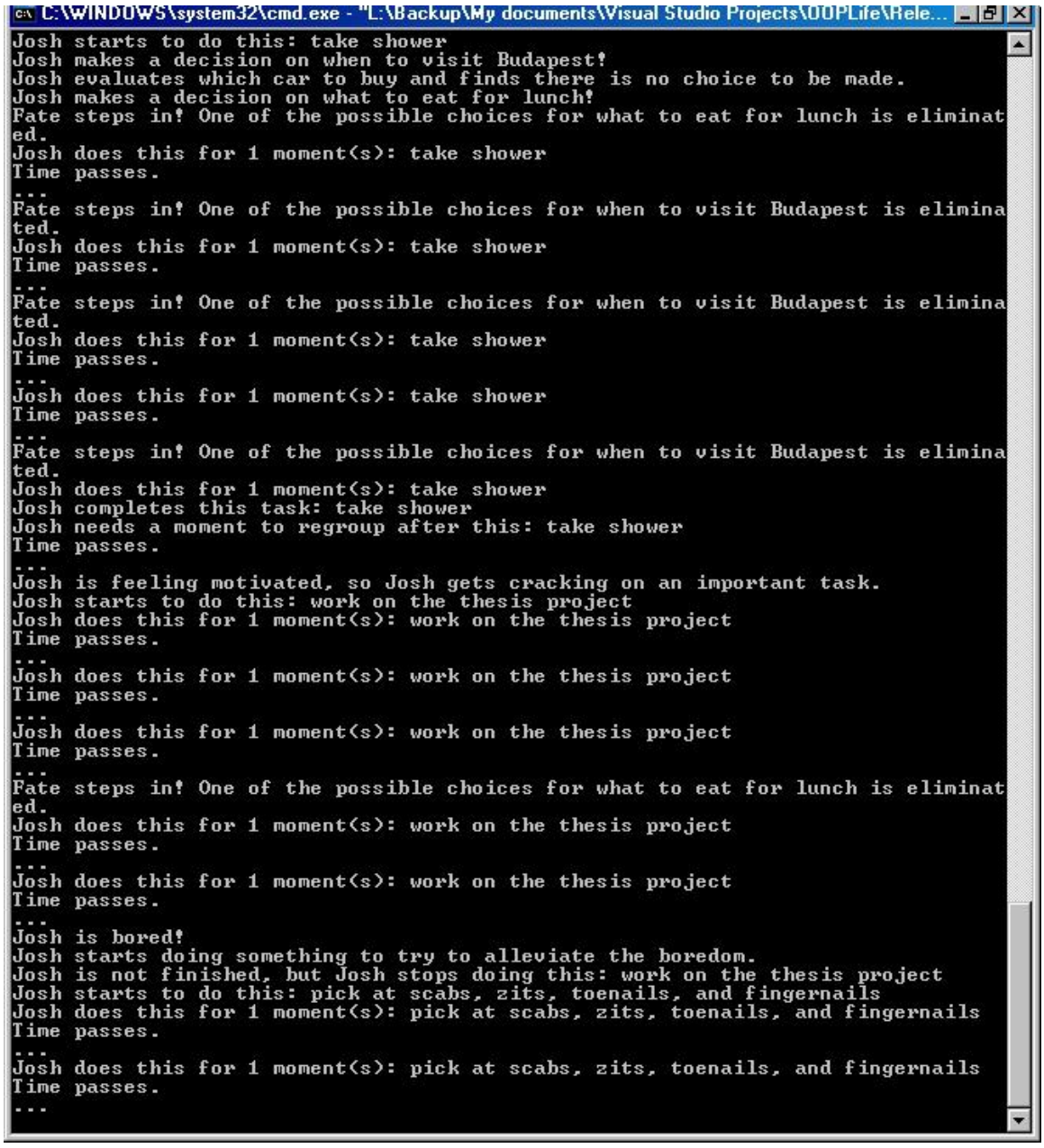


Fig.16
Joshua Samberg, Self SDK. Output of the previously pictured program.

[generative software art]

The artwork is created by the use of a generative system, which sets a few basic rules to define the initial conditions on which the generative process is based. It is self-contained and operates with some degree of autonomy.
It offers a powerful and fascinating tool for the artists as it allows them, in a sense, to create their own 'worlds' which function according to the rules chosen by the artists.
The idea of automating language, cognitive reasoning and art existed already in the 17th century. For example in 1674, Quirinus Kuhlmann, follower of the earlier mentioned Ramon Lull, published in a book 'Epistolae Duae' a discussion about automatically generated art and its cognitive limitations. (according to Cramer 2005 : 105).
A later example of an already functional self-operating generative algorithm is cellular Automaton (CA) first considered by John von Neumann in the 1940s, and then becoming well known in the 1970s after the 'Game of Life', devised by the British mathematician J.H.Conway, was published.
Game of life "is run by placing a number of filled cells on a two-dimensional grid. Each generation then switches cells on or off depending on the state of the cells that surround it."(Worlfram MathWorld) Even though the rules defining the behaviour of the cells are very simple, they produce amazingly complex results. (Fig 17).

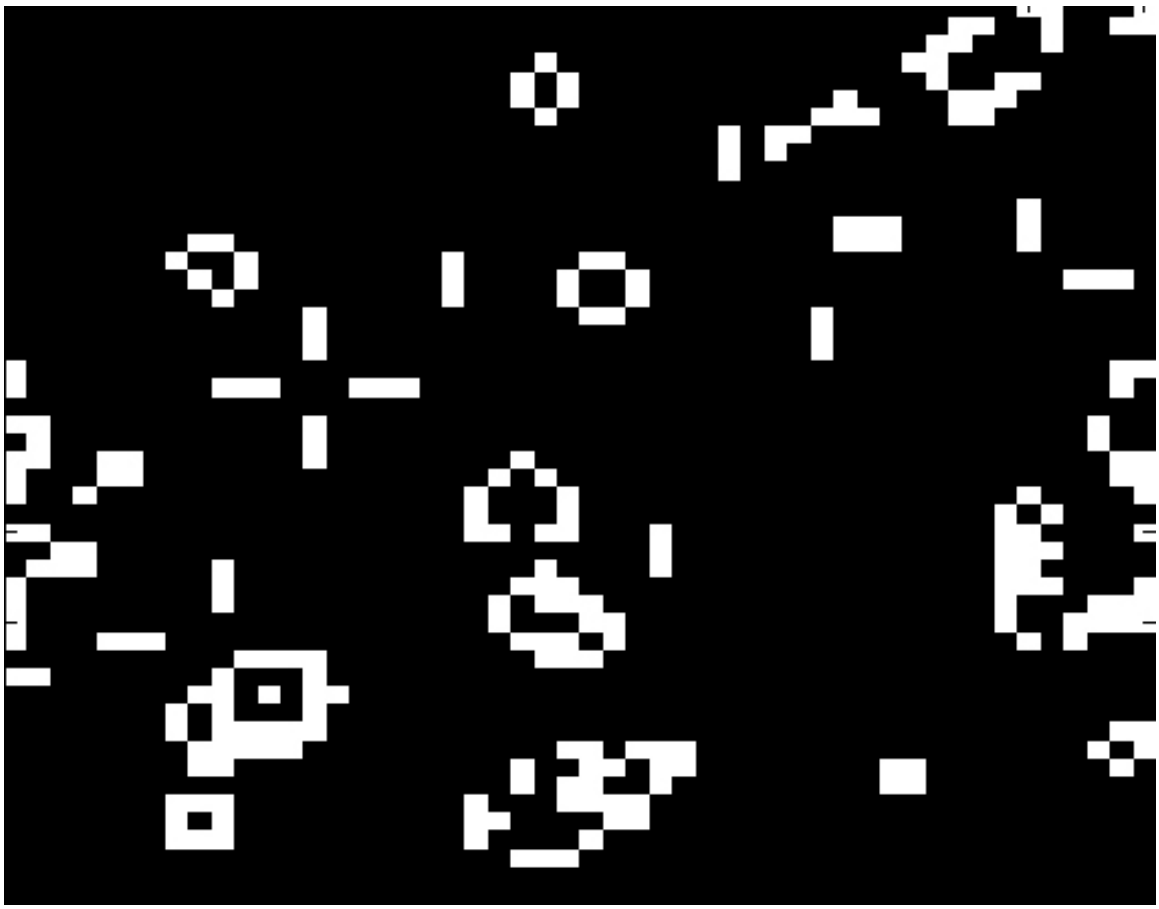


Fig.17 Celullar Automaton ‘Game of life’

> “ The rules are defined as follows. All eight of the cells surrounding the current one are checked to see if they are on or not. Any cells that are on are counted, and this count is then used to determine what will happen to the current cell.
>
> 1. Death: if the count is less than 2 or greater than 3, the current cell is switched off.
>
> 2. Survival: if (a) the count is exactly 2, or (b) the count is exactly 3 and the current cell is on, the current cell is left unchanged.
>
> 3. Birth: if the current cell is off and the count is exactly 3, the current cell is switched on. “ (Worlfram MathWorld)

Many art and scientific works have been inspired by Conway’s work. 'The simple rules which produce complex results’ is a significant feature of self- generated algorithms.

[augmented [software] reality]

If software has to be executed by the computer in order to be defined as software is a topic of ongoing discussions. In my opinion, programming code executed by human can clearly be defined as a software art.

The '.walk', entitled "no software but walkware"(runme.org, 2003), a psychogeographical game played in various locations of the world, can serve as a brilliant example of such artwork.

> "A generative psychogeographical walk only has a direction but no destination. There are no good or bad results, only the results you come back with. That's why failure is impossible. On the other hand it must always fail because the city never fully complies with the demands of the algorithm. But it doesn't matter. What is important is to practise this art of being in between." (Crystalpunk, 2006):

.walk can have simple rules, such as in the following example:

"programming .walk for dummies
// Classic.walk

Repeat

{

1 st street left
2 nd street right
2 nd street left

}"
(Crystalpunk, 2006)

This example shows simple, 'classic' psychogeoraphical algorithm. We may have a look at other, more complex instructions:

"/ Fibonacci .walk
// 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Z = 1
Z(x) = 0

```
Repeat

{
Z Left or right {random}

Z(y) = Z
Z = Z + Z(x)
Z(x) = Z(y)

}
```
”(Crystalpunk, 2006)

Different from the previous example, here the choice of direction is not strictly set, but depends on the decision of the individual following the algorithm. The code also computes its own next turn according to the Fibonacci number series, which is infinite. “…following this .walk applet to its logical conclusions must soon becomes surrealistic, if not downright absurd.”(Crystalpunk, 2006)
To follow similar instructions obviously requires some mathematical and programming knowledge, the terms ‘software’ and ‘programming’ are not used only as metaphor.

[software net-art]

If software art is defined as “art of which the material is software.”(Cramer&Gabriel,2001:3), definition of net-art as software-art becomes problematic.
The common use of the term ‘Net-Art’ refers to the media by which the work is distributed, not to the material from which it is created. As defined by pioneers of net.art Natalie Bookchin and Alexei Shulgin: “net.art […] describes an art and communications activity on the internet.”(Bookchin & Shulgin, 1999)
HTML (Hypertext Markup Language), extensively used on the World Wide Web is a markup language, not a programming language. HTML documents are not software! To make web pages interactive, programming code can be embedded in HTML. But for example, applets (programs) written in Java programming language can be ‘called’ by the HTML document, but the code is then, same as

other Java programs, executed by JVM (java virtual machine), utility which is installed on the particular computer.

Another possible approach would be to consider software artworks which use the structure and content of the Internet (WWW) as a topic, to be the ‘software net-art’. It can perhaps also include artworks interested in the social behaviour on the internet. Various internet visualising projects would then belong to this category.

Conclusion

Software art is a unique artform. Even though it is novel its conceptual roots reach far back to the history of humanity and its very nature is a manifestation of the fundamental laws of the Universe.
It is unfortunate that such a powerful creative tool, as the computer, is treated in the way that its full potential can not be revealed. As John Maeda states:

> "Imagine if the computer we touch everyday were viewable through some special glasses that reveal this alternative reality. We would see something like a shimmering material of pure electric thought, perhaps incomprehensible but at least several universes away from the dreary click, keypress, and drag that we associate with modern computing."(Maeda, 2000 : 59)

Actually, we would not need the special glasses but new patterns of understanding in order to comprehend the fascinating realm of shimmering electric thought! Necessity of the new ways of apprehension is not limited to the subject of computational art, but like anytime when the 'change of paradigm' happened, these need to emerge in all fields of human knowledge.
We may look forward to the moment when the general way of understanding, currently still influenced by Cartesian philosophy will change, and when looking at computational artwork, we will be able to perceive more than images moving on a flat computer screen.

Bibliography

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

Ars Electronica Archive, 2003,Code The Language of Our Time, catalogue 2003 http://www.aec.at/en/archives/festival_archive/festival_catalogs/festival_catalog.asp?iProjectID=12281

Ashvaghosha, The Awaking of Faith, translation D.T Suzuki,1900,Open Court, Chicago

Bateson Gregory, 1979, Mind and Nature; A necessary Unity, Hampton Press, Cresskill, NJ published 2002

Blumenthal David R., 1978, Creator and Computer in *Understanding Jewish Mysticism* ,p 22-29, New York: Ktav Publishing. Currently http://www.js.emory.edu/BLUMENTHAL/CreatorandComputer.html

Bookchin Natalie and Shulgin Alexei , 1999, Introduction to net.art (1994-1999), http://www.easylife.org/netart/

Capra Fritjof, 1975, The Tao of Physics, Willwood House, UK

Capra Fritjof, 2002, The Hidden Connections, published by flamingo, London, 2003

Cramer Florian, 2001, Digital Code and Literary Text, http://www.netzliteratur.net/cramer/digital_code_and_literary_text.html

Cramer Florian and Gabriel Ulrike, 2001, Software Art and writing http://plaintext.cc:70/essays/software_art_and_writing

Camer Florian, 2002, Concepts, Notation, Software, Art, http://cramer.plaintext.cc:70/essays/concept_notations_software_art

Cramer Florian , 2003, Exe.cut[up]able statements:The Insistence of Code, http://plaintext.cc:70/essays/executable_statements

Cramer Florian, 2005, Words Made Flash http://pzwart.wdka.hro.nl/mdr/research/fcramer/wordsmadeflesh/

Crystalpunk, 2006 http://www.socialfiction.org/index.php?search=walk

Dale Nell & Weems Chip, 2005, Programming and Problem Solving with C++, 4th edition, Jones and Bartlett Publishers International

Emoto Masaru, 2005, The Hidden Messages in Water, Simon & Schuster

Encyclopedia Judaica, 1971, Keter Publishing House Ltd., Jerusalem

Flusser Vilem, 1973 – 1991, Writings, edited by Shtrohl Andreas, 2002, University of Minnesota Press,MN

Heisenberg W., 1963, Physics and Philosophy, Allen&Unwin, London

Howe Denis, 1993, Free On-line Dictionary of Computing
http://foldoc.org/

Jewish Virtual Library
http://www.jewishvirtuallibrary.org/index.html

Maeda John, 2000, Maeda & Media, Thames&Hudson Ltd, London

Maeda John, 2004, Creative code, Thames&Hudson Ltd, London

Murti T.R.V., !955, The Central Philosophy of Buddhism , Allen&Unwin, London

Reas Cassey, 2003, Programming Media, in Ars Electronica Archive 2003
http://www.aec.at/en/archives/festival_archive/festival_catalogs/festival_catalog.asp?iProjectID=12281

Reas Casey & Fry Ben, 2007, Processing : a programming handbook for visual designers and artists , MIT Press

runme.org - say it with software art! , 2003
http://www.runme.org/project/+dot-walk/

Samberg Joshua, 2006, Self SDK,
http://iceberg901.com/selfsdk.html

Sepher Yetzirah, translation by W.W.Wescott, 1887
http://www.sacred-texts.com/jud/yetzirah.htm

Scholem Gershom, 1971, The Mesianic Idea in Judaism and Other Essays on Jewish Spirituality, Schocken Books Inc.

Stocker Gerfried, 2003, Code - The language of Our Time, in Ars Electronica Archive 2003

http://www.aec.at/en/archives/festival_archive/festival_catalogs/festival_catalog.asp?iProjectID=12281

Wolfram Mathworld
http://mathworld.wolfram.com/