10º ENCONTRO INTERNACIONAL DE GRUPOS DE PESQUISA
convergências entre arte, ciência, tecnologias & realidades mistas

»_ell_ Worl_!« - the rule becomes the author.

Christian Heck
Kunsthochschule für Medien Köln.

Ting Chun Liu
Kunsthochschule für Medien Köln.

Mattis Kuhn
Kunsthochschule für Medien Köln.

Tiago Ive Rubini
Kunsthochschule für Medien Köln / cAt, GIIP – UNESP.

Julia Nakotte
Kunsthochschule für Medien Köln.

RESUMO

O presente texto é uma reflexão sobre o processo de escrever linhas de código como forma de produzir literatura. Trataremos de uma *esolang* específica chamada *Beatnik*. Uma esolang é uma de linguagem criada especificamente para se ter uma ideia de como funcionam as linguagens de programação, sendo comum em processos artísticos e criativos que lidam com código, poesia e práticas semelhantes. Beatnik foi criada por Cliff Biffle no início da última década, e serviu de inspiração para escrever nossa própria linguagem de programação, também chamada Beatnik, projetada principalmente por Ting Chun Liu e Christian Heck. No decorrer do texto, discutiremos tal processo. Primeiro, falamos sobre a estrutura do código, que funciona através de *stack piles* e o processamento de palavras em inglês e alemão. Também nos interessa falar sobre autoria na perspectiva de redes sociotécnicas, conforme proposto por Bruno Latour. Finalmente, iremos relacionar o trabalho de escrever através de algoritmos à literatura moderna, como nos trabalhos da geração *Beat* e de Gertrude Stein.

Palavras-chave: arte e tecnologia, programação, código aberto, esolang, literatura.

ABSTRACT

The text is a reflection about the process of writing code as a form of literature. It deals with a specific esolang *called* Beatnik. *An esolang is a language system composed only for the sake of seeing how it works, which is usual for artistic and creative processes dealing with coding, poetry and related practices. Beatnik was created by Cliff Biffle in the beginning of the last decade, and it served as an inspiration to write our own programming language, also called Beatnik, designed mainly by Ting Chung Liu and Christian Heck. In the paper, we will discuss how this came to be. First, we talk about the structure of the code, that works through stack piles and the processing of words in English and German. We are also interested in talking about authorship through the perspective of sociotechnical networks, as pointed out by Bruno Latour. Finally, we relate our will to write through algorithms to modern literature, such as the works of the Beat generation and Gertrude Stein.*

Keywords: art and technology, programming, open source, esolang, literature.

1. Introduction: authorship, inorganic actors and esolang.

In his text *The Cathedral and the Bazaar*, Eric Raymond talks about an unorthodox form of authorship, which he calls “constructive laziness”(RAYMOND, 1999). By this, the author and

programmer means that “good programmers know what to write. Great ones know what to rewrite (and reuse)”, shifting the value of good coding from trying to be a pioneer to putting effort in practical solutions. Raymond does not argue in favor of disrespecting one’s authorship, but assumes that knowledge itself claims to be further developed, not by an individual but by collective work.

If we take this idea in a more radical approach, considering a given text or writing process as a sociotechnical network, as Bruno Latour (2005) points out in his Actor-Network Theory, this sort of collective practice could be extended even to non-human, inorganic actors. Algorithms are not authors in the traditional sense, since they do not have the same agency as humans, but they definitely influence the way we write. That was exactly the inspiration for this text.

The questions to what extent “_ell_ Worl_!” can be considered literature or poetry and to whom authorship can be attributed are closely related to the algorithmic approach. In addition to the basic conceptual idea, it is above all the firmly defined rules for generating the text, the algorithms as actors, and the use of foreign texts as a basis that determine the “new” text. However these factors collide with notions of authenticity and creativity, which still play a relevant role in the classification of literary works today.

Beatnik, the basis of the following artistic process and text, is a simple stack-based esoteric programming language written by Cliff L. Biffle in 20011. It was adapted in the making of the poem »_ell_ Worl_!«, bringing together the poem and its Beatnik-Interpreter, and now taking part in the present paper. If we say “stack-based” we talk about the way in which Beatnik functions. Beatnik is a Stack Machine, a machine which operates stacks, i.e. an ordered pile of items where the addition of new items and the removal of existing ones always takes place at the top of the pile. In the case of Beatnik a stack represents a single sequence of objects and/or values that are modified by simple instructions. “Stack-based esoteric programming language” means not so much that Beatnik is a language but rather a form of language encoding.

Even though esoteric programming languages (esolangs for short) are called esoteric, they actually have nothing to do with esotericism. The crucial thing about esolangs is their relation to conventional programming languages. They are usually not designed for practical use, but remain in part as a theory or idea, sometimes even without having found a real implementation

in technology. They can have academic value, but often they are simply a joke or a play on words to test a concrete idea. Therefore, these languages are perhaps not always directly useful, but nonetheless are very good testing grounds for new artificial, artistic, formal and natural language concepts, since they can be designed with the idea of cultural and artistic value in mind.

2. Variables and operations.

Value assignment is the most elementary concept in programming. In Beatnik, a value is assigned to the individual tokens (in our case, words). Overall, these values can be seen as references. In other words, as signs or variables. Variables, which store information that can be used and/or changed in your program.

Below in this paper we will see functions that generate lists of words so that they can operate within the Beatnik stacking machine. In all examples we work with the higher programming language Python, a formal language frequently used in our context for processing natural language texts. There are many ways to process and generate natural language texts in systems and machines et vice versa.

We are standing right in front of the fourth wave of the COVID-19 pandemic, caused by the SARS-CoV-2 Delta variant. For about a year now, our everyday life is more than ever guided by collected data, measured data, metadata, personal data, movement patterns, behavior patterns, speech and thought patterns & customs, and many other sorts of formalization of ourselves. To some extent even vaccines work algorithmically. Through vaccination we insert pieces of text with code function into our bodies or a certain muscle regions. To do so we implemented ourselves into machines, making ourselves machine readable, so to speak, and embedded in generalized spaces which can be divided in *syntactic spaces*, *semantic spaces* and in *pragmatic spaces*2.

Those pieces of text, representing parts of ourselves, create patterns of meaning inscribed into machine-generated vector spaces. And those spaces increasingly influence our everyday lives. Some of those spaces are epistemological, others can be called cognitive systems. They are abstracted actors changing our perspectives about the material effects of the pandemic, helping us see them in a bigger picture, or a network. They help us see what we cannot see with our

biological eyes alone and assist us in navigating through complex, written, network-like interrelationships such as dataflow graphs, diagrams and heat maps. Places become datapoints and spaces movement-profiles.

Those little helpers support us right now in this mental performance, in our human ability to abstract concrete things so that we can estimate the consequences for our own actions in the here and now. They enable us to behave responsibly towards our fellow human beings, to be social so to speak. To be allowed to live a life according to our social principles of action: “Act only according to that maxim by which you can at the same time want it to become a general law” (KANT and KEHRBACH, 1788).

In summary, we can say: we write texts into machines and machine-interpreters which help us to see, sometimes through diagrams, sometimes through maps, sometimes through new generated texts, and when we read them and organize our daily life through them, then, exactly through this action we generate together with our little helpers new meaningful words, and ways, and cultural spaces, and also cultural values.

3. A code-literary historiography - from Beatnik to Gertrude Stein.

The term “Beatnik” comes from the name given to the authors of a literal avantgarde movement in the New York literary scene, namely the Beat generation who were also called The Beatniks.

One of the most important works of Beatnik-literature (besides Allen Ginsberg’s *The Howl* (GINSBERG, 1956) and William S. Burroughs *Naked Lunch* (BURROUGHS, 2013), first published in 1959) is Jack Kerouac’s road trip novel *On the Road* (KEROUAC, 2008) first published in 1957. It was Kerouac who introduced the term Beat Generation to the New York literary scene.

In *On the Road*, Kerouac writes about the aforementioned authors and many others, such as the time he spent with Neil Cassady. Kerouac wrote the complete draft of the aforementioned in three weeks in April 1951 in an apartment on West Twentieth Street in Manhattan (KEROUAC, 2008). This document was written as one long paragraph on eight long sheets of tracing paper, which he later glued together to form a 120-foot scroll, creating one of the most significant and provocative artifacts in contemporary North-American literary history.

Most Beatnik authors did not really care whether their works referred to canonical concepts of literature. They rejected any form of definition of their activity – they were not worried about being seen as legitimate literature makers or not. Also, they refuted the concept that writing is a clear and fixed function of authors. For them, life and writing can be seen as one, and the focus of their writing and living was the present. Their aim was to create an aesthetics of existence based on a different perception of the present. They used language in an experimental way, inscribing it with a different rhythm, sound, and dynamic; namely the *Beat*.

An endless search for possibilities to fathom the limits of language and to decipher the preconditions of understanding, precisely because the dominant language is the language of the dominant class. That means, an undermining of a dominant-legitimate use of language must be formulated through the use of alternative language techniques.

William S. Burroughs and Brion Gysin, for example, created step-by-step instructions to grammatically scramble the social structure of their time, thus deciphering the prevailing *syntax of control*.

Often the Beatniks were called *those who write in rhythm*. They wrote like they walked and walked like they wrote. Finding the rhythm of the city, the rhythm of the machine. Like Gertrude Stein, they poetically appropriated the language technologies of their time.

Stein was a writer who, beyond question, experimentally discovered spaces in her literature to replace the outdated systems of the 19th century, stepping towards a new understanding of language. Parts of her works in Poetry belongs to an early type of combinatory poetry. We can see those combinatory plays with language especially in works like *Tender Buttons* (STEIN, 1991). She used formalized restrictions and self-given constraints to process language, filter it and transform it mainly on a syntactical level into poetry.

We can see this exemplarily in Steins Essay *Poetry & Grammar* (1935), where she wrote: “I like the feeling the everlasting feeling of sentences as they diagram themselves.”

In this literary journey from prose to poetry, she was “trying to understand (…) to feel inside the words that come out to be outside of you” and she tried that experiment by using an approach to diagram sentences. So she began to enumerate all those words which do something, because

in her eyes, “as long as anything does something, it stays alive”. She was doing something close to what nowadays in Computerlinguistics we call POS-Tagging:

```
('they', 'PRP'),
 ('see', 'VBP'),
 ('that', 'IN'),
 ('darker', 'NN'),
 ('makes', 'VBZ'),
 ('it', 'PRP'),
 ('be', 'VB'),
 ('a', 'DT'),
 ('color', 'NN'),
 ('white', 'JJ'),
 ('for', 'IN'),
 ('me', 'PRP'),
```

Figure 1: POS-Tagging example. Source: https://dev.ground-zero.khm.de/ell-worl-paper/4literary.html . Accessed on 10.11.21 .

A grammatical classification.

If we define a text as a combination of elements (characters, words, lemma, interpunctuation, POS, n-gram, etc.), we can count these elements. We see that in first instance in the way of expanding old syntaxes and playing with them as a new grammatical order.

In a way, we can say, that the syntax of our formal technical languages (programming languages and so forth…) went hand-in-hand with early poetic language techniques and experiments. Especially the culture of writing esolangs is a very code-poetic way of keeping this spirit alive.

4. How to code Beatnik.

Cliff Biffle, who wrote the original Beatnik esolang, says in his website3 that “Beatnik is a very easy language to learn: it has a small command set, a very relaxed syntax, and you can find a reference to the vocabulary in any toy shop”.

During our dive into Beatnik-Poetry Ting Chun Liu created a Python package as well as a Github Repository based on Cliff Biffle’s Beatnik, turning it into functions understandable by writers inexperienced with coding.

Starting with the primary interaction mechanism, the code is progressively modified to meet individual author needs. By getting familiar with the logic of Beatnik’s operation, we can influence the way we write and purpose a new writing system.

For more information on how to access the source codes and program using Beatnik, it is possible to visit the executable version of this paper and get more specific instructions[4]. Below, we will present some of the structural characteristics of the code.

4.1. Constructing a Beatnik Library.

It is important to see the programming language Beatnik not as a simple text translation, but as an interface that breaks down the original text into digits - in this case integers - and then uses those digits to generate text again.

In Beatnik, words have a different meaning than in our everyday use of words. The program will articulate them to reveal the inner structure of its technical environment. In Beatnik, *words* mean actions that are performed. The actions are linked to words by means of letter values, i.e. free from the actual meaning of the words (For example: “Kunstwerk” has the same meaning as “Berechenbar”). In certain instances, they will reveal in their use the close connection between the signs in the machines and all the abstract intermediate levels that lie between the words and our thinking.

In order to compile a dictionary, an adequate corpus of texts is required. In its quality as well as in its quantity. This short sample text is used as a model to show step by step how to convert text into a practical reference for the use of Beatnik. Therefore, we put the string *loud exhibition sometime tells photograph to print only realities* into the variable *text*:

```
text  = '''loud exhibition sometime tells photograph to print only realities'''
```

Figure 2: example of string in variable. Source: https://dev.ground-zero.khm.de/ell-worl-paper/4literary.html . Accessed on 10.11.21 .

Constructing the Beatnik library consists of three key steps: **Tokenization**, **Scrabble**, and **Stacking**. Each step has its necessity: *tokenization* converts text to data, *scrabble* transforms data to value, and finally, the value is output as a character based on the ASCII table through a *stack* machine5.

4.2. Beatnik Programming.

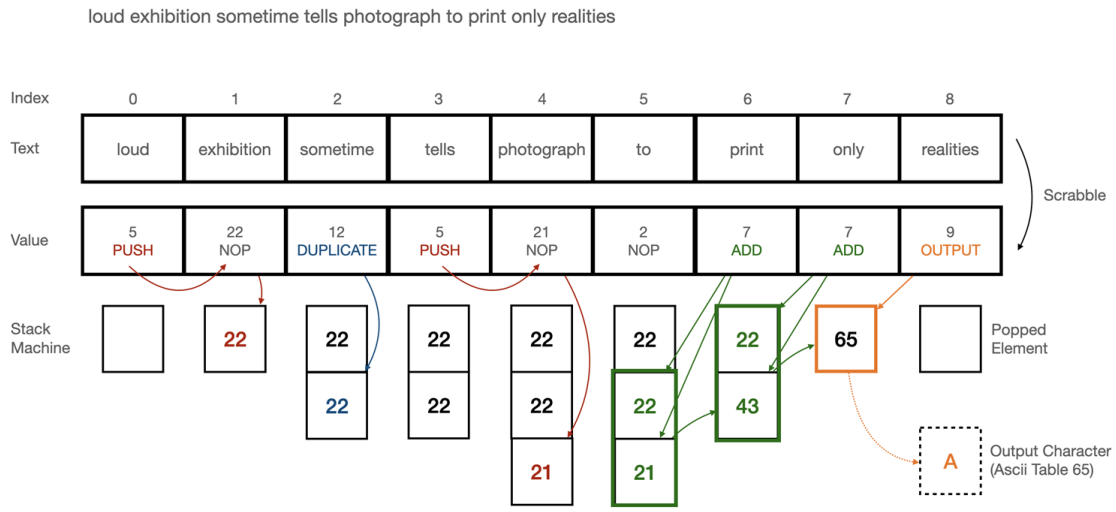

Figure 3: visualization of how the code works, image created by Ting Chun Liu.

To traverse the string *loud exhibition sometime tells photograph to print only realities*, we start by initializing the index to 0, which indicates the first number 5, which corresponds to the function *PUSH* as we have seen above, in the stack.

We move to the next element in the list by increasing the index by 1, based on the word *exhibition* , the value accordingly is 22. After inserting it, we now have [22] on the stack.

The next value 12 indicates the action *Duplicate*: the first element of the stack is popped out, and inserted onto the stack twice. The stack machine we now have contains two elements [22,22].

Then we *PUSH* (Function 5) the value 21 from *photograph* onto the stack, and ignore the next element 2, which does nothing. There are two 7 from the words *print,only* in the list following. The function corresponding to 7 is *ADD*, which means to insert the sum of the last two elements into the stack machine. Sum of 22 and 21 is 43 , sum of 43 and 22 is 65.

Finally, the value from the word realities is 9, which means *OUTPUT*. We then look for the top element [65], translate it with the ASCII table, and output it as character *A*.

By stacking more text and using functions to add and subtract the sum of values, we can output more characters by matching the obtained values to the ASCII table.

5. Esolangs.

Of course, there are many more literary esolangs out there then Beatnik. You can find a huge list of esolangs in the esoteric programming languages wiki6. In our context Shakespeare Programming Language (SPL), by Karl Hasselström 7, also stack-based, is one worth mentioning. The code of the infamous Hello World Program starts like this in SPL:

```
Romeo, a young man with a remarkable patience.
 Juliet, a likewise young woman of remarkable grace.
 Ophelia, a remarkable woman much in dispute with Hamlet.
 Hamlet, the flatterer of Andersen Insulting A/S.

                    Act I: Hamlet's insults and flattery.

                    Scene I: The insulting of Romeo.

and so on...
```

Figure 4: excerpt of Hello World Program in SPL. Source: https://exmediawiki.khm.de/index.php/Shakespeare_Programming_Language . Accessed on 10.11.21 .

The program follows the typical structure of a drama: title, *dramatis personae* (variable declarations), acts / scenes (instructions). The output function, which is in Beatnik the scrabble-word-value 9 (f.ex. the word *realities*) is in SPL *Open your heart* or *Speak your mind*.

Another stack-based esoteric programming language is *Piet8*. A language in which programs look like abstract paintings. It uses 20 colours, of which 18 are related cyclically through a lightness cycle and a hue cycle. A single stack is used for data storage, together with some unusual operations.

In Piet, a Hello World Program could result in possibilities as such:

Figures 5 and 6: examples of applications with the Piet esolang. Source: https://dev.ground-zero.khm.de/ell-worl-paper/4literary.html#id9 . Accessed on 10.11.21 .

And of course, it could it be different again. In 2001 Gerson Kurz, a well known inventor of several esolangs, wrote *.Gertrude9*, which consists of sentences in any natural language. For each sentence, the average length of its words is calculated (rounded off), and then the number of words longer than the average is divided by the number of words shorter than the average. The resulting fractions denote instructions and operands. Sentences that contain ! or ? are ignored. In the same way that in the preparatory phase of writing »ell Worl_!«, where the Codichter*innen Julia Nakotte, Julia Vergazova, Ting Chun Liu, Patricia Falk, Julia Maja Funke, Lisa James, Lukas Ortheil and Tom Tautorus took works from Charles Bukowski, Georges Perec, Walter Benjamin, and Herman Melville (amongst others), Gerson Kurz took *Tender Buttons* by Stein and Kants *Critique of Pure Reason* as a base to write a Hello World Program in a different syntactical order.

```
Winged to be winged means that white is yellow and pieces pieces that are
brown are dust color if dust is washed off then it is choice that is to say
it is fitting cigarettes sooner than paper.
A CARAFE THAT IS A BLIND GLASS.
Suppose they are put together suppose that there is an interruption
supposing that beginning again they are not changed as to position suppose
all this and suppose that any five two of whom are not separating suppose
that the five are not consumed.
Cut more than any other and show it.
He would have been peculiarly well fitted to give a truly scientific
character to metaphysical studies had it occurred to him to prepare the
field by a criticism of the organum that is of pure reason itself.
The difference is spreading.
A sight a whole sight and a little groan grinding makes a trimming such a
sweet singing trimming and a red thing not a round thing but a white thing a
red thing and a white thing.
```

Figure 6: Hello World in .Gertrude esolang. Source: https://dev.ground-zero.khm.de/ell-worl-paper/4literary.html#id42 . Accessed on 10.11.21 .

The version of Hello World in .Gertrude which came out from it can be read as above (the first 16 lines from 270).

6. Intermezzo: Beatnik as a Live Coding Language.

> "in programming languages […] their different structures — semantic descriptors, grammar and style in which algorithms can be expressed — lend themselves not only to different problem sets, but also to different styles of thinking" (CRAMER, 2008, p. 170).

What might live coding look like in relation to poetry? The practice of live coding is known from the fields of sound and video, which is almost its sole use. Typically, functions are written to create media in real time and to modulate media over time: changing frequency, velocity, transformations of objects or colour values, etc., often in iterations of a loop (a beat). Sound

and image become analog again after synthesis, while text (which was digital before computers) remains digital. While a pitch or colour value may change in steps imperceptible to us, quasi-analog, in text there is a jump - to the next character or string. (BARTHES, 2019, p. 114) In this sense, text is more digital than sound and image. Of course, this is not an obstacle to live coding poetry, it just makes it harder. Since in live coding it is usually not only the result but also the code that is of interest (and presented to the audience), one possibility in relation to text could be to interpret natural language as code – as is executed in the esoteric programming language Beatnik. As Beatnik is inspired by the Beat Generation, which focused on the present time, it seems to fit very well to performing practice, which happens in present time. The following are considerations for why Beatnik is suitable for live coding (even if, or perhaps because, it is challenging).

Live coding is a performance practice. In the performing arts, there are traditional notions (albeit temporary) of good performative quality. This is especially true in music. Quality is achieved by practicing and thus "mastering" an instrument. Even though this is only one and not necessarily the decisive category in the evaluation of a performance as such, it will be explored further here. In concrete terms, this means: How can the (technical) quality of the performer be measured in live coding with Beatnik?

The first level is the code in a human interpretation. One of many possibilities to generate a l is the number sequence *5 22 5 5 7 12 12 7 7 7 9*, which we get for example by the code line *eH rzauHaods fo ew gonstr lfolow ertabica akslw by wont uyg alpce.* But also through

eH rauosadHz of we ngtors flwloo ieratcab lkasw yb ntow ygu clpea. de. Or like Julia Vergazova has done in coding the letter l for *_ell_ Worl_!*: *He Hazardous of we strong follow bacteria walks by town guy place*.

Since in Beatnik's interpretation of the code only the sum of the values of the characters of a word is relevant and not their order, in terms of the operations triggered the three codes above are the same for the machine interpreter. In human interpretation, they are not. While the first two seem rather arbitrary, the last sequence suggests a non-arbitrary design and encourages us to interpret it rather than the others. Of course, we also expect meaningful text in the output produced by the code, perhaps even text that cannot only be interpreted by us, but is itself

(meaningful) Beatnik code again. (Of course, as every text written in the Latin alphabet is Beatnik code, every output produced by Beatnik is a possible input again.)

Another level for measuring quality are the instructions generated by the code. As a reminder, the ASCII value for *l* is *108*, so to generate a *l*, the value *108* must be at the top of the stack and output using the numeric value 9 (in the example above by the word/ command "place"). The easiest way is to generate the required number with the simplest possible arithmetic operations (add, duplicate) and then translate this into a desired character via output. Further, more fastidious possibilities exist, such as for example using the stack as data structure, thus storing values purposefully for later use or swapping them. Another challenge is the integration of jumps into the program flow. Thus the quality of a performance shows up on three levels: the program in human-readable form, the operations executed by this code and the output produced. As a further quality the changeability of the program is added: e.g. to produce large differences by the addition or alteration of some characters at the program code, or the opposite, to produce no differences at all at the output. Since the order of the letters within a word is irrelevant, with identical characters different (meaningful) words could be generated, for example "town" and "wont". But also many different programs could be written from scratch, which all generate the same output. Both are possibilities that run over several iterations of the program, so the Beatnik performance would also follow the scheme of the loop, as is usually the case for live coding with the media of music and images.

From the point of view of the prerequisites of the programming language, Beatnik lends itself to live coding poetry because, on the one hand, it works with natural language on two different levels and, on the other hand, quality characteristics of the code can be derived from the design of the language itself. Suppose we set material or tool mastery as a quality criterion. Beatnik requires parallel thinking at the levels of (1) input, (2) operations effected by it, (3) data in the stack, and (4) output. This is a demanding, purely mental task, so the question is whether it could be done by a machine. It seems so. But Beatnik comes with a kind of a loophole, which makes it difficult for a machine, but instead offers another possibility for the performer: an additional input prompt, which is triggered with the value 8. Through this the performer can insert code into the Beatnik interpreter, which is not part of the (more static) regular input code.

For e.g., combined with a loop of input prompts (through skip back functions) poetry inside poetry could be written.

We have seen some challenges of the language that make it suitable for live coding performances. But ultimately the challenge of the language is only part of the game; perhaps what is more important is the influence that the specifications of the language and, for example, the restriction to small changes in iterations for the purpose of live performance have on the resulting texts.

7. Conclusion.

Pop art, Fluxus, Conceptual art & literature, Concrete poetry, Cybernetic art, Appropriation art, Net art & literature, the Lettrists, Glitch art, Code poetry, Dadaist poetry, Experimental literature, Code art & literature, don't know where to start and where to end…. All of these isms, genres and avantgarde movements in art and literature have one thing in common: they propose that the central element in cultural work and practices like art and in literature can be the conceptual, or the algorithmic work. That means, working with instructions for action that come close to the performative element of the code (BAHJOR and BAUER, 2016). They all start from the idea that rules can specify processes for the production of art, or as the conceptual artist and minimalist Sol LeWitt put it: "The idea becomes a machine that makes the art"10.

> "If one wants to understand how art and technology relate to each other in our European tradition, one has to go far afield. Today we are used to see intuition and rationality as opposites. The common origin of poetics (poietike - the creative, poetic art) and technology (techné) in Greek poiesis, on the other hand, has been largely forgotten."[11]

Today, the way we make art and literature goes closer and closer back to its roots. It has changed fundamentally in the last two centuries. Especially in the last two decades of this new millennium, our handling of text, our reading and writing behaviour has moved significantly closer to that of the machine. Some of us perceive this consciously, for others this kind of writing is already part of the new normality.

Interfaces like Flickr, Instagram, Facebook are forcing us to a pre-defined use of language, for example in Twitter a very condensed language (<140 char). The use of hashtags makes it easier for us to get lost in #-tag clicks like a lettrist derivé (GOLDSMITH, 2011). We dig and search through text material, through clicks and search functions, trying not to get lost in jungles of

web pages reading hypertext, and writing new cryptic passwords and passphrases every week. Most of these texts are not being read in a linear fashion. Most of them are not even meant for linear reading, or even being read by humans at all! Most of the time we regard them as visual patterns and then in turn consciously use this experience in our work and daily lives.

We continue to experiment with yet undisclosed spaces in order to assemble the fragmentary into poetic arrangements, to replace the outdated systems of the 20th century, stepping towards a new understanding of language, … somehow like Gertrude Stein did a hundred years ago.

REFERÊNCIAS BIBLIOGRÁFICAS

BAHJOR, Hannes and BAUER, René. **Code und Konzept: Literatur und das Digitale.** Reihe Generator. Frohmann, Berlin, 2016.

BARTHES, Roland. **Über mich selbst.** Matthes & Seitz, Berlin, 2019.

BURROUGHS, William S.. **Naked lunch: the restored text.** Grove Press, New York, 2013.

CRAMER, Florian. **Language.** In Matthew Fuller, editor, **Software Studies. A Lexicon**, pages 168–173. MIT Press, Cambridge, Massachusetts, 2008.

GINSBERG, Allen. **Howl and other poems.** San Francisco, 1956.

GOLDSMITH, Kenneth. **Uncreative writing: managing language in the digital age.** Columbia University Press, New York, 2011.

KANT, Immanuel and KEHRBACH, Karl. **Kritik Der Praktischen Vernunft.** Reclam, Leipzig, 1788.

KEROUAC, Jack. **On the road: the original scroll.** Modern classics. Penguin, London, 2008.

LATOUR, Bruno. **Reassembling the social: an introeuction to the Actor-Network Theory.** Oxford University Press, 2005.

RAYMOND, Eric S.. **The cathedral and the bazaar.** Knowledge, Technology & Policy, 12(3):23–49, 1999. Publisher: Springer.

STEIN, Gertrude. **Poetry and Grammar.** Lectures in America, 1935.

_____. **Tender buttons: objects, food, rooms**. Number 8 in Sun & Moon classics. Sun & Moon, Los Angeles, 1. publ. in pap edition, 1991.

Ting Chun Liu

Born in Taipei, Taiwan, Ting Chun Liu is a postgraduate student at the Academy of Media Arts Cologne, Germany. Graduated from Taipei National University of the Arts. His works revolve around the cognitive and bodily relationship between reality and digital, discoursing and blur the digital dualism in the Internet era through audio-visual, network and interactive programming.

Christian Heck

Christian Heck currently works at the Academy of Media Arts Cologne (KHM) as assistant prof for aesthetics and new technologies. He teaches there in the field of "Experimental Informatics", where he is also doing his PhD under Prof. Dr. Georg Trogemann. He is member of the Computer Scientists Forum for Peace and Social Responsibility (FifF), glitch artist, cypherpunk and code poet.

Tiago Ive Rubini

Tiago Ive Rubini is a doctoral student at São Paulo State University Júlio de Mesquita Filho (UNESP), and a fellow and guest PhD student at the Academy of Media Arts Cologne (KHM). They participate in the groups cAt, GIIP (UNESP) and exMedia Lab (KHM). Also they work with sound art, performance and have an interest in open source culture and intersectional activism and research.

Julia Nakotte

Julia Nakotte is a postgraduate student at the Academy of Media Arts Cologne (KHM). Her texts are mostly created with a conceptual background. Through text-generating algorithms and interactive computer game poetry, she questions the role of authorship today.

---

[1] URL: http://cliffle.com/esoterica/beatnik/ (Accessed 24.08.2021).

[2] Respectively: the syntax of a language (a system of signs) describes the rules according to which its contructs are formed. The syntax is the study of word formation and sentence structure; the semantics of a language describes the meaning of its constructs, in other words, it's the doctrine of the words' meanings and the content meanings of a language; pragmatics is giving a sense and a meaning to the series of signs into our daily life through action.

[3] See note 1.

[4] More details in https://dev.ground-zero.khm.de/ell-worl-paper/4literary.html#id7 (Acessed 10.11.21).

[5] See more details about each step on the same link as note above.

[6] URL: https://esolangs.org/wiki/Main_Page (Accessed 10.11.21).

[7] URL: http://shakespearelang.sourceforge.net/report/shakespeare/shakespeare.html (Accessed 24.08.2021).

[8] URL: https://www.dangermouse.net/esoteric/piet.html (Accessed 24.08.2021).

[9] URL: https://exmediawiki.khm.de/exmediawiki/index.php/.Gertrude (Accessed 24.08.2021).

[10] URL: https://theoria.art-zoo.com/paragraphs-on-conceptual-art-sol-lewitt/ (Accessed 24.08.2021).

[11] URL: https://georgtrogemann.de/2016/07/comment-on-ralf-baecker-the-paradox-of-knowing-universals/ (Accessed 24.08.2021)