Beyond the Code/Executable Dualism

Once the participant is immersed in the Virtual Reality or Metaverse game-space or is experiencing the Augmented Reality flesh-of-the-world which is hybrid of what was previously called digital-virtual and what was previously called quark-atomic-physical, then both superficial and deep changes to the code must be possible and available while inside that *Star Trek*-like "Holodeck." The world which is neither natural nor artificial comes into mature existence when this hybrid constitution is achieved through sharing between humans and the world, or between technology and nature. We need new understanding and designing of the game universe, beyond the duality of programmer and user. We dream of a single-electron transistor, a latency-free network, a cosmos where we can make up the rules, where one is free to devise any laws for the invented universe that one wishes. Rather than replacing or escaping from physical reality, this new software will enhance the *Wirklichkeit* (substantiality, actuality) of the experimental ecological neo-habitat.

D. Fox Harrell's Phantasmal Media

In his book *Phantasmal Media: An Approach to Imagination, Computation, and Expression* (2013), D. Fox Harrell strives to establish a new relationship between the human or posthuman imagination and computing.[731] Writing code, or working actively with computational media, are, for Harrell, activities of artistic, cultural, social, critical, and personally empowering expression. The great expressive potential of computational media comes from their capability to both reveal and construct what Harrell calls "phantasms." Phantasms concretize cultural ideas as imaginative sensory artifices. Computational media are especially adept at detailing, fleshing out, and codifying cultural ideas. Phantasms are subjective cognitive phenomena which are situated, distributed, and embodied. They are combinations of mental imagery and collective ideology. Harrell classifies phantasms as: senses of self, metaphors, social categories, narratives, poetic thinking. Developers of computing systems – working with images, text, sound, video, animations, and other computer-based media, both expose oppressive phantasms which perpetuate power relations and create new empowering phantasms.

Phantasms that can be created with computers combine sensory imagery with conceptual ideas. They encapsulate beliefs, knowledge, social problems, the encounter between self and others, and experiences of everyday life. Cognitive processes bring together "epistemic spaces" and "image spaces." "Image spaces give phantasms salience and sensory structures," writes Harrell.[732] He confronts the crucial question of how computers can be deployed expressively. We are only beginning to understand "expressive epistemologies." These are human worldview-based data structures that enable digitalized imaginative worlds and poetic phantasms. Phantasms are involved in apprehending the world on levels ranging from simple events to complicated artworks. On the socially critical side, computational phantasms expose the largely fictional nature of social norms which reproduce power and oppression. Power relations can be as "real" as the fascist boot stomping on your face, but the existentialist philosophical position is that the first step towards overthrowing power relations is to stop internalizing their self-justifying narratives, and instead deconstruct them, in your own mind.

The practice of making effective phantasms involves skillfulness in the translation between subjective or cultural constructions of meaning and the data structures and algorithms of computer science. Harrell divides the knowledge field of creating compelling expressions with software code and artistic digital design tools into (1) Subjective Computing (creative, poetic, figurative, and ethical/political expressions that resonates with the imaginative experiences of users), (2) Cultural Computing (Subjective Computing grounded in cultural context), and (3) Critical Computing (raising Cultural Computing to the level of confronting social phenomena and bringing about societal change).

A cultural phantasm is a group-shared phantasm that can be described according to a comparative, descriptive, or computational cultural model. Cultural phantasms tend to be socially entrenched to the point that we are often not aware of them. Computing systems can be designed to render cultural phantasms more visible.

Critical Computing is the design of computing systems done while contemplating the social and political values that they embody. "Agency play" is the expressive personal and social impact of interactive systems, while combining user and system agencies.

Computational creativity can bring to life the form of imagination that Harrell calls the "poetic phantasm": impactful mental imagery and ideas involving verse, metaphor, allegory, or narratives. What he calls "expressive epistemologies" are especially inspiring or evocative cultural productions such as artworks. "Polymorphic poetics" are, for Harrell, aesthetically rich structural mappings in systems and interfaces among goals, designs, and significations. Subjective, Cultural, and Critical computing systems stimulate and disseminate phantasms.[733]

Harrell seems unclear on the question of whether the decisive level affecting if a given technology empowers or disempowers people is the design or use of the technology. He writes:

> The values built into the structures of computer systems can serve to either empower or disempower people. The same technologies that allow one to chat with a loved one across an ocean in a different country, or that customize a user interface based on where one lives, can be used for illegal surveillance and restriction of privacy. The

> same technologies that can be used for educational training or artful entertainment can be used for online bullying…[734]

From the first sentence to the second sentence of this passage, Harrell contradicts himself in a way that indicates that he has not sufficiently thought this issue through. Which is it? Are these values "built into the structures" of systems in their design at a fundamental level (my position) or are the values a matter of how the already-designed-and-structured technology gets used?

The substantial value of Harrell's work for the present study is his manifesto-like advocacy of the integration of "humanities and arts-based approaches to critical engagement with society and the world" into computer science. He draws attention to the gap between computational media artistic expression and "more mature media forms that have much more established conventions and strong communities" engaged in creativity and theorizing. Yet I wish to point out another undifferentiated blurry area of Harrell's research. He writes:

> Computational media systems all too often remain focused on self-reflexive exploration of the media themselves, as opposed to producing transformative content.[735]

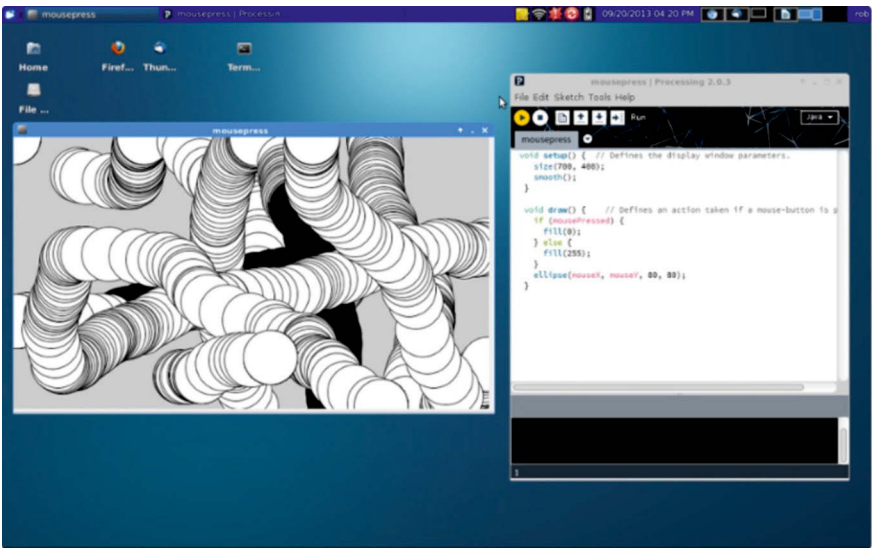
Are these two activities – "self-reflexive exploration of the media themselves" and "producing transformative content" – really in such strict opposition to each other? Should not the theory and practice of social transformation also include reflection on the nature and form of the media itself that is deployed to attempt to bring about social change? The more salient dividing line – between two conceptual sub-categories of the category of "self-reflexive exploration of the media themselves" – would be between works which are merely fascinated with stretching the technical possibilities of given software environments (thus indistinguishable from standard Silicon Valley practice) and those which reflect on the digital media technology in ways informed by philosophical or art- or media-theoretical questions about form and the underlying human-machine relationship established by the media at hand.

Casey Reas and Ben Fry's *Processing* Language

Casey Reas and Ben Fry are the inventors of the *Processing* programming language and interactive development environment. *Processing* is an example of a relatively simple object-oriented language to be used by artists, designers, and other creatives to make generative art projects and interactive graphics. *Processing* is based on the more sophisticated Java programming language. In their book *Make: Getting Started with Processing, A Hands-On Introduction to Interactive Graphics*, Reas and Fry achieve more than teaching the reader Creative Coding on a practical level with hands-on example programs called "sketches."[736] They bring the reader along chapter-by-chapter into a deepening understanding of computer language concepts which correspond to the decade-by-decade history of programming language paradigms as the successive innovations of functions (1970s), event-driven programming (1980s), and the classes and software objects of ob-

ject-orientation (1990s) were introduced. We practice the pedagogical method of double-learning of the technical patterns of software code and the philosophical-cultural patterns of some of the successive (past, present, and future) historical phases of informatics. At each step of the learning process, we become more proficient as *Processing* Creative Coders and gain cultural science understanding.

Learning any programming language begins with the "Hello World" program. As Wendy Chun points out, this simple introductory iteration makes sense to the novice and is readable.[737] It consists of a series of declarations and imperatives. It produces immediate results (two words get displayed in an app or a console) and hints subtly that all code will instantly do something palpable and immediately verifiable. We learn that computer programming started historically with machine languages, then was followed by assembly languages, and then higher-level languages. A typical program in the 1960s (to continue my simplified decade-by-decade history of programming paradigms) was the "spaghetti code" of a series of sequential instructions issued in linear fashion inside a single "main" procedure. Similar to the single main() function, coding in *Processing* begins with the setup() and draw() functions. In setup() is the code that is executed one single time at the startup of the application. The draw() function contains the code that handles what the software will do in response to user interaction events. We learn how to do both direct and algorithmic-generative visual drawing.



Casey Reas and Ben Fry, Processing Integrated Development Environment

With the 1970s innovation of functions, program control could be delegated to a helper routine. The code became more modular, reusable, and efficient. One goal was to reduce the writing of duplicate code. In Chapter Nine of the book, Reas and Fry introduce the artefact of functions which attained prominence with the C programming

language of AT&T Bell Labs in the early 1970s. An input-output dialog takes places between the "calling" and the "called" function. The "calling" function can pass data via parameters to the "called" function to which it temporarily hands over control. When the called function is finished, it sends control back to the caller, as well as a return value as its output. Note that code and data are still strictly separate from each other. In this paradigm, technology is a tool used instrumentally by the human-subject-who-is- in-charge that acts on some non-living object.

In event-driven programming, explained by Reas and Fry in Chapter Four on Response, the program is no longer proactively calculating something or sending instructions to processor. The software sits there passively in a draw() loop the code of which gets executed 60 times per second while waiting reactively for a user input event – via mouse, keyboard, microphone, or camera – to occur. This is a step beyond the programmer-as-subject ruling over the machine-as-object model. It is a step towards the software as autonomous and semi-alive.

As explained in Chapter Ten on Classes and Objects, the object-oriented Object is a complex data type composed of many values of many variables which are grouped together. Software classes are defined either as built into the language, made available for use in third-party libraries, or designed and written by the programmer herself. The software class is the specification, and the software object is a single runtime instance of the class. The class encapsulates the values of the properties (fields) of an object and the operations (methods) on that object into a single "object-oriented" unified entity. Data and code are unified. The software object is, in this programming paradigm, on its way to becoming autonomous and self-aware. An object has introspection: it knows both its internal data and actions on itself.

Oliver Bown on Computational Creativity

In his book *Beyond the Creative Species: Making Machines That Make Art and Music* (2021), Oliver Bown systematically considers the field of computational creativity.[738] This includes examining creative Artificial Intelligence and Deep Learning neural nets, the automation of creative tasks by machines, and the situation of human artists and designers working in partnership with intelligent machines. Bown also investigates the impact of generative software environments and technologies on the creation of music, visual art, stories, poems, and games. There is a marked difference between systems of autonomous computational creative agents and the use of advanced AI techniques as support tools by a human artist.

The study of creativity touches upon perennial questions about emotions, beauty, the sublime, culture, interpersonal relations and individual experience and psychobiography. Bown also thinks about the consequences of these human and machinic modulations for the so-called "creative industries." He argues that, to fathom computational creativity, we should not only ponder computationally creative algorithms, but must engage with creative artistic activity. Bown explores the psychology of creativity and how it may synergize with algorithmic automation. His work is at the meeting point between interaction design and complex systems.

Bown sees the study of computational creativity as a transdisciplinary endeavor. It requires a multiplicity of approaches ranging from anthropology and cognitive neuroscience to design, art theory, philosophy, and creative practice research. It encompasses the sciences of evolutionary biology, Artificial Life, and AI. Within conventional computer science, it involves the sub-fields of algorithms, human-computer interaction, and user experience.

Bown cites the idea of the avant-garde experimental musician Brian Eno that listeners will soon embrace self-evolving generative compositions as preferable to fixed compositions. Eno thinks of himself not as a human composer composing a single composition to be heard over and over, but rather as creating a system that, in turn, composes an ever-changing composition that is entirely flexibly and spontaneously varying. Musical patterns will be generated on the fly spawned by an algorithm and will respond to the events of mood or user input. Music will be parameterized – just like generative software or interactive visual artworks.

In their installations, new media artists make use of computers, electronics, video, Internet (net.art), telerobotics, telematic networks, remote telepresence, mechanical engineering, bionics, and transgenics. Interactive participatory works and environments invite the user to discover "polysensoriality." The perceptual-motoric-tactile dimension of embodiment is restored to equal standing with the symbolic-rational dimension emphasized by much of traditional art. The artist who utilizes information technologies designs an open-ended work the trajectory and outcome of which are not predefined by the artist, but which rather depend to a great degree, and in a "post-humanist" way, on both the actions of the human participant-user and the "semi-living entity" which is the generative intelligence of the work.

Walter M. Elsasser and the Trans-Computational

Much of the current prevailing biological paradigm reduces understanding of the living organism to the combinatorial model or formula of the genetic code. But the genetic message is only a signifier of the complete reproductive process. "The message of the genetic code," writes Walter M. Elsasser in *Reflections on a Theory of Organisms: Holism in Biology*, "does not amount to a complete and exhaustive information sequence that would be sufficient to reconstruct the new organism on the basis of coded data alone."[739] This reductionism on the part of biologists corresponds to the prevailing computational paradigm of binary or digital computing of the twentieth century. It is almost as if, according to Elsasser, biologists decided, since this is the limit of the computing power that we have, we will devise a biology that functions within the restrictions of what we can compute.

The question is: how to handle complexity. In conventional computer programming, this is handled essentially with the Cartesian or top-down Method – break down the complex problem into smaller, more manageable parts or sub-problems. But it is impossible to apply the Cartesian Method to, for example, quantum-mechanical (quantum physics) generalized complementarities like the wave-particle duality or the Heisenberg Uncertainty Principle. Whereas the top-down method may work for mechanical systems, it

cannot be of much use when we aspire to the understanding or creation of something that is living or semi-living. The approach that I propose is to identify relationships of similarity, to find samples or patterns that capture something of the vitality and complexity of the whole without breaking it down in a reductionist way. This resembles the "perceptrons" approach pioneered by Frank Rosenblatt.[740]

According to Elsasser, we need Holistic Biology where the living organism in its full complexity is considered. The structural complexity of even a single living cell is "transcomputational." Elsasser writes that the computational problem of grasping a living organism (or organic structure) is a problem of unfathomable complexity. The single living cell is involved in a network of relationships with all of life. The individual member of a species decodes in real-time, in each new circumstance, its species-memory. It creatively retrieves this species-memory through a process of information transfer that is effectively "invisible," and does not take place via any intermediate storage or physical transmission media. This holistic information transfer happens over space and time, "without there being any intervening medium or process that carries the information."[741] Whereas the genetic code is memory as "homogeneous replication," holistic memory, for Elsasser, is one of "heterogeneous reproduction."

Elsasser's interrogation of how we could consider organic life as an information system and his ideas about the trans-computational, invisible data transfers, and the logic of similarity or resemblance are useful notions for thinking about a possible paradigm shift towards a post-combinatorial computer science.