What Am I Looking At?: An Approach to Describing the Projected Image in Live-Coding Performance

Shawn Lawson
Rensselaer Polytechnic Institute
lawsos2@rpi.edu

Ryan Ross Smith
Monash University
ryanrosssmith@gmail.com

ABSTRACT

The authors investigate and deconstruct a live-coding performance's projected image to develop an approach for describing what is within that projected frame. The work of Roland Barthes, Michel Foucault, and Vilém Flusser serve as models for constructing a descriptive framework based on text, graphic, annotation, interface, and image.

1 Introduction

In this paper we are focusing on the projected image commonly found in live-coding performance and developing an approach for how to discuss what is seen within that visual and performative frame. We develop an understanding of the objective differences between textual and graphical elements. This approach includes the defining of several terms: Image, Text, Graphic, Annotation, Interface, and several hybrid combinations, upon which we will postulate how to classify what it is we see and what it might mean, if anything.

Live-coding is defined as writing portions of a program while it is running (Ward et al. 2004). For our purposes we contextualize this in a performance setting where the screen of the live-coding performer is projected for the audience to see. While the projected image is not a requirement for a live-coded performance, the projection of one's code is certainly a hallmark of contemporary live-coding practices.

Using live-coding's projected image, we analyze three perspectives of text and image; include Roland Barthes's *connotation* and *denotation*, Michel Foucault's unraveled calligram, and Vilém Flusser's technical images. Through these different viewpoints we derive a method of categorically describing the contents of a live-coded projected image. Finally we employ this descriptive method with some examples of live coding to demonstrate the method's possible utilization.

2 Image

The first term we consider is Image, which we define to be the projected image as the container for all visual elements in a live-coding performance. There has been research on the perception of and response to the live-coder's projected image with various constituent groups (Burland and McLean 2016; zmölnig 2016; Rodríguez and Rodríguez 2015; Roberts 2016), the aesthetics of code (Cox, McLean, and Adrian 2004; Cox and McLean 2013), the semiotic meaning of live-coding and sound (Sorensen, Swift, and Riddell 2014), as well as a critique of the practice of projecting live-coding as a whole:

> This tradition of projecting screens is itself open to criticism; the audience members may feel distracted, or perhaps even excluded by the projection of code written in a language they do not necessarily understand (McLean et al. 2010, 1).

For our purposes, the Image does not include anything beyond the boundaries of the projection, meaning that the performer, their computer, and anything else is superfluous. It is also important to note that in this context there is no Image if there is no projection. Clearly this is not meant to suggest that a projection-less, live-coding performance is inferior, but that without the projected image, or Null-Image, there is no container for which the constituent elements of the live-coder's screen can be assessed.

3 Text

One of the primary elements found in a live-coding performance Image is the code or Text content. This Text content of the live-coder's projected Image has been addressed by several authors, including arguments for live-coding as a type of scoring (Magnusson 2014), live-coding as information or annotation (Roberts 2016; McLean et al. 2010), and the use of an intertextuality of puns or meta-narratives (Rodríguez and Rodríguez 2015, @Herrera_Machuca_2016).

IOhannes m. zmölnig uses a four-layer approach to describing the perception of code: graphics, glyphs, text, and instructions and algorithms. The first layer, graphics, refers to the visuality or overall graphic quality that code has (2016). In similar fashion, Rodríguez & Rodríguez have also talked about the visuality of code/text:

> The projection of the code transforms into a complex language to communicate something to the interface, as well as a piece of a visual section that interacts with the spectator. … to construct and make possible the interaction between a range of different texts, perceiving text as an image … (2015, 1).

But this interaction between the code and the spectator may take on different forms based on the spectator's understanding of the code. Rodríguez & Rodríguez further developed a framework of this relationship in order to identify different models of understanding that range from completely uninitiated to the expert live-coding practitioner:

> These kind[s] of practices can not be perceive[d] with a simplistic point of view of a work of art that can be [sold] or that can be held in a museum o[r] gallery context. So, these practices do not need somebody, as a curator[,] but a community that works as a medium o[r] learning process to create not just readers but producers (2015, 4).

How poorly or well a viewer comprehends the code presented in the Image is, as Rodríguez & Rodríguez suggested, dependent on how well initiated the viewer is regarding live-coding practices, and specifically, how well the viewer understands the functionality of the code. It is here that we can begin to define our second and third terms: Text, where Text refers to the code, and Text-Image, where Text is perceived as an Image.

Beyond contemporary, live-coding-specific analyses of live-coding's Image there are more general approaches to unpacking the Image. One method is to understand the relationship between the Image and the Text in the context of visual rhetorics.

4 Rhetoric of the Image

Visual rhetoric is similar to that of textual literacy, where someone is able to analyze and evaluate text; in this case, the analysis and evaluation is of images. When we discuss visual rhetorics, we are interrogating the message an image may contain. This relates to visual culture, and how individuals interpret what they see.

Roland Barthes was the first to look at how the intermingling of text and image had the potential to represent complex interpretations. In his Book *Image - Music - Text*, Barthes describes a method for talking about the interplay of text and image together, specifically in regards to pre-computed images: drawing, painting, and chemical photography. In regards to his critique of press photography he uses two primary terms that build on Saussure's semiotic definitions of signifier and signified:

Denotation: A sign turning into its literal, first order meaning. For example, an image of a bunny signifies a bunny; the signifier.

Connotation: A sign turning into its associated, second order meanings, often associated with personal history and cultural specificity. For example, an image of a bunny may signify softness, cuteness, an inclination for aggressive reproductivity, or the Easter holiday; the signified.

For Barthes, what the image depicts becomes a denoted meaning, while the process of creation, the creator's decisions, and material become the connoted message. The connoted message often relies on a common cultural history or knowledge base to be clearly read. For example, a black and white, closely cropped photo of a dusty farmer in a field wiping sweat from their brow might connote the impression of a stoic, hard-working, and possibly poverty-stricken individual. That same photo in brilliant color looking upward at the farmer might connote entrepreneurship, success, and have aspirational vision. With both images the denoted message is simply a farmer.

From a historical perspective, image and text combinations were used as illustration: Text would provide the direct literal meaning while the image would illustrate a connotated meaning. Regarding the introduction of photography and captions, Barthes states the following: “Firstly, the text constitutes a parasitic message designed to connote the image, to ‘quicken’ it with one or more second-order signifieds” (1977, 25). When the image was already intended to connote a message, the caption now steam-rolls it with it’s literal connotation. Barthes is arguing that text-captions are now secondary to the primary message of the image, as these text-captions forcefully push the connotation forward. Prior to the photograph, images would help to depict or document what was being stated in a text. With the photograph, someone could have an image of *exactly* what they wanted. If we reconsider our farmer, then a caption may state our previously considered connotation: This is a hard-working farmer living a quiet, stoic life. Barthes believes this caption steals the impact of the image by helping when help is not needed.

Barthes continues, “Secondly, the effect of connotation probably differs according to the way in which the text is presented. The closer the text to the image, the less it seems to connote it; caught as it were in the iconographic message ...” (1977, 26). This implies that as the text is moved closer to, or even superimposed upon the image, it effectively merges into the image and loses most of its connotation:

> Here text (most often a snatch of dialogue) and image stand in a complementary relationship; the words, in the same way as the images, are fragments of a more general syntagm and the unity of the message is realized at a higher level, that of the story, the anecdote, the diegesis ... (Barthes and Heath 1977, 41).

This indicates that when text and image are layered together they may create a more holistic message than if they are placed further apart. We see this occurring in earlier forms of text and image mergers, for example, with illuminated manuscripts where the first letter of a page or paragraph is also an image, see figure 1[1]. Here we see that the text, a letter, is the image, and the image is the text.

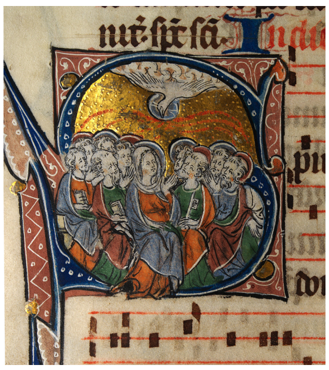


Figure 1: **The Pentacost, from an illuminated Catholic liturgical manuscript, c1310-1320**.

In the next section we step back slightly and refocus on Text as the only component of an Image, because Text is typically the primary component of the live-coder’s projected Image.

5 Text-Image and Live-Coding

In a live-coding performance, how is the Text in the Image functioning? Is the Text a caption? What are the denotative and connotative meanings? These questions are particularly difficult to answer in a situation where the primary visual element is the Text itself, and, as alluded to previously, is further complicated based on whether or not the Text can be interpreted by the viewer. Zmölnig speculates about this issue with regards to who is technically capable of comprehending the Text, which in this case is the code:

> However, having the source code available will only help those that are able to read and interpret it. In this context ‘code’ is not only denotational ‘instruction code’ to be executed by the machine, but becomes connotational ‘secret code’ that can only be deciphered by an initiated minority (2016, 207).

The initiated minority here are the programmers who are able to read and decipher the code. Zmölnig implies with this statement that only live-coders can interpret the connotative meaning. This appears an unfair presumption to the uninitiated, who may develop their own valid connotative meanings. In this situation of live-coding, Barthes would say that the Text denotes exactly what it is: an algorithm, description, or score. The connotations might be tech-y, future-y,

[1] https://en.wikipedia.org/wiki/Illuminated_manuscript

or even voyeuristic in the moving image context of watching someone edit. Even if the uninitiated may not be able to decipher the 'secret code,' for them, the code may be perceived as unreadable text, potentially becoming purely a Text-Image. Is this also the case for the programmers if they aren't actively following and interpreting the Text (code); does the Text become a Text-Image for them?

This flip-flopping of Text and Text-Image has been explored in other conceptual forms, like the calligram, which we now turn to with Michel Foucault and René Magritte.

6 The Calligram

Michel Foucault writes in *This is not a pipe* that the calligram is text written or drawn in a shape where both the text is readable and the shape is recognizable. Any related interplay of text to shape is defined by the creator of the calligram, although often the intent is that there is a symbiotic relationship between the two that enhances reflexive and recursive reading and viewing. See figure 2[2] for an example calligram.

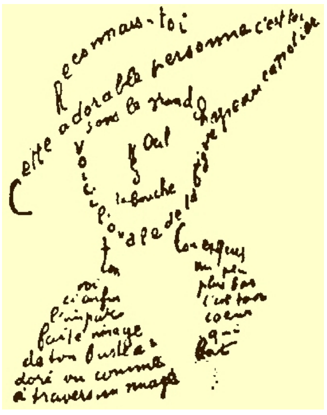

Reconnais-toi
Cette adorable personne c'est toi
sous le grand chapeau canotier
Oeil
la bouche
Voici l'ovale de ta figure
ton cou exquis
Et voici enfin l'imparfaite image de ton buste adoré vu comme à travers un nuage
un peu plus bas c'est ton coeur qui bat

Figure 2: **Calligram by Guillaume Aplollinaire, date estimated early 20th century**.

Foucault takes an interest in the calligram format as it relates to René Magritte's well known work *La trahison des images [Ceci n'est pas une pipe]* (The Treachery of Images [This is Not a Pipe]), 1929. His argument is that *La trahison des images* is a preserved, unraveled calligram (1983).

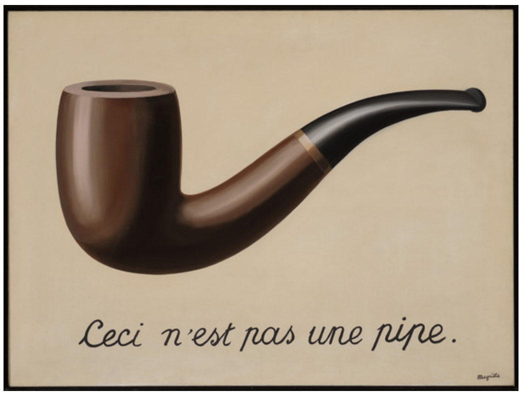


Figure 3: **René Magritte's *La trahison des image,* (1929)**[3].

To begin, Foucault declares that the calligram has three roles, "... to augment the alphabet, to repeat something without the aid of rhetoric, to trap things in a double cipher" (1983, 20). Meaning that letters are augmented to become drawn elements to comprise a shape. Then that the emergent image is created from the text and no other text is provided to describe or as Barthes would say, connote the image. Lastly, for the calligram to work, the letters as image and image as text must exist, with each encoded in the other. When looking at Magritte's painting, Foucault uses these three roles, and states that Magritte preserves them in an unraveled way.

Unraveling step one is extracting the Text from the Text-Image while retaining the Graphic of the pipe. The letters of Magritte's calligram are removed from their shape (Text-Image) and placed in a linear line of Text, as a caption, to explain

[2]https://en.wikipedia.org/wiki/Calligrams
[3]https://collections.lacma.org/node/239578

or connote. However, Foucault is clear to note that the Text is within the same frame as the Graphic of the pipe (both should be considered part of the complete Image). He even goes so far to say, "... the represented pipe is drawn by the same hand with the same pen as the letters of the text ..." (1983, 23) indicating that both Text and Graphic have the same degree of hierarchical visual prominence.

Unraveling step two is to deny the message repetition. The Graphic of the pipe is denied its first order denotation by the extracted Text from it's prior Text-Image. The Graphic is a pipe, but when the Text performs, now, as a caption, it states that the Graphic is not a pipe. The Graphic seen and the Text read now deny each other.

Unraveling step three is to release the double cipher with the purposeful ambiguity of "Ceci" (This). Does "Ceci" mean the phrase "Ceci n'est pas une pipe" or the Graphic representation of the pipe? Foucault asks us to consider three cases:

- (Graphic of pipe) is not a pipe
- This is not a (Graphic of pipe)
- This (Graphic of pipe and the phrase "This is not a pipe") is not a (mixed combination of Graphic of pipe and phrase "This is not a pipe")

In each of these propositions the encoding or cipher of letters as image and image as text fails to complete. Foucault explains that the white space (separation) between the Text and the Graphic (the emptied Text-Image) reject their ability to encode each other.

If, for a thought experiment, we re-raveled Magritte's painting so that the Text once again filled the Graphic to become the Text-Image we may have the following, see figure 4, a complete calligram.

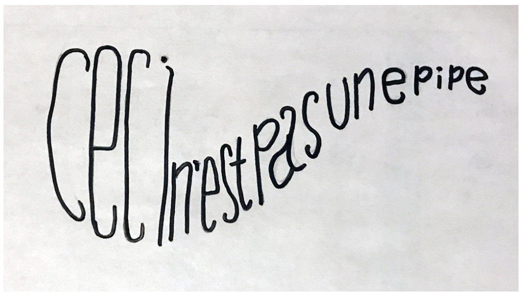


Figure 4: **An attempt at re-raveling Magritte's *La trahison des image* as per Foucault.**

7 Calligrams and Live-Coding

Using Foucault's ideas consider the following example in the Python programming language as a calligram. Also, we refer to the executed output: Text, Graphics, Sound, or error as the shape of the calligram.

```
this is not "a pipe"
```

Starting with the augmentation of alphabet, this straw-man example works as a calligram because the letters are also functioning as executable, logical code. That shape as an executable logical proposition is below.

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'this' is not defined
```

Second, the tautology test holds, because the Text and the shape are saying the same thing. The Text is stating that "this is not 'a pipe' "; and, the shape is a "NameError: name 'this' is not defined." There is a space here at which one could argue that in the Text, it is true that we don't really know what the variable 'this' is, which is supported by the above error. If we execute the following

```
"this" is not "a pipe"
```

The result evaluates to True as 'this' is a string and the logical test can execute. We know what 'this' is - it is the string "this". We know that "this" is not "a pipe" as does the interpreter. If we had previously defined 'this' to be something, for example 'null', the statement would evaluate False, which we would also know.

Lastly, to complete the double cipher the text and shape must intertwine and encode each other. On the surface this seems impossible: how does the output become encoded by the input and vice-versa? We must consider part three as unraveled, with the Text and the output as separate and not intertwined. If we want to think about them being encoded within each other, then it may be possible to think of both the Text and the shape (executed output) as representations of a single collection of bits. Those bits when interpreted as Text give us the code on screen, and those exact same bits interpreted to be executed code give us the output result. When perceived through this lens, then the double cipher requirement holds.

Realistically this feels slightly unsatisfying to explain Text-Images, because now our Text-Image has morphed into the shape of a calligram which is defined as the executed output. Whereas previously our Text-Image was defined as Text that was not readable and comprehend-able code, acting in itself as a purely visual element.

At this point we re-consider our mangling of the double cipher as encoded bits with the help of Vilém Flusser through his description of particles and technical images.

8 The Technical Image

Media theorist, Vilém Flusser, looked beyond traditional media in his book *The Universe of Technical Images* to consider digitization and computation when developing his concept of the technical image. Technical images, as described by Flusser, are entirely different from traditional images and cannot be understood with traditional methods:

> The difference between traditional and technical images, then, would be this: the first are observations of objects, the second computations of concepts (2011, 10).

What he's saying is that drawing, painting, photography and so on are forms of representation based on the observation of something that exists, whereas, technical images are ideas visualized through algorithms, data, and computation. In *Chromatic Algorithms*, Carolyn Kane decodes in more detail how this differentiation works:

> Technical Images are a priori abstractions only later made concrete. They are distinguished from "traditional images," which are used to grasp or "depict" the world and the environment through "magical" actions, which are translated onto a surface (2014, 228).

To clarify, the use of "magical" refers to Flusser's use of the word to describe the interplay of signifier and signified. Here we see a reference to Barthes's description of images with denoted and connoted messages, which as we indicated is built on Saussure's signifier and signified terms. Kane is establishing that the prior semiotic methods of image rhetorics is different from the methods needed for technical images. She continues this line of thought by saying that Flusser's technical images are post-hermeneutic (after or without meaning, reference, history, or capable of being interpreted) and cannot be observed, which describes one of Flusser's primary tenets (2014, 228).

Kane quotes Flusser arguing that technical images must be known and decoded in terms of how the technical image is created, but not what the technical image means; "The semantic and pragmatic dimensions of technical images are identical" (2014, 228; 2011, 49). This means that technical images have no sign, therefore no signifier nor signified. The sign (the object in the world that is to be represented by observation) does not exist except by concept alone. Another way to think about this is to say that the *what* of the image is the same as the *how*.

A similar approach is taken by Zabat Patterson in his book *Peripheral Vision*, in which he uses the term "diagram" by Wolfgang Lefevre and Gilles Châlet, which is traced from a history of Architecture; "In other words, the diagram both questions and uncovers the process by which ideas are formed in the mind" (2015, 21). This appears very similar to technical images in that they are both emergent from concepts or ideas. Patterson further supports Flusser's definition of the technical image as not meaningfully interpretable or post-hermeneutic:

> The diagram, on the other hand, is privileged as a figure of use and transformation — not a representation of reality, but something more akin to a stand-in. The diagram — as a mode of representational practice — marks a shift from a concern with the perceptual qualities of the object to a concern with it's undergirding conceptual relationships. No longer mimesis — but rather system (2015, 22).

Patterson is clear to state that diagrams are not representations from observation, but that they are from concepts and systems where systems could be an algorithms, data, and computation - the *how*, which is as we learned is also the *what*.

In summary, traditional images are depicted representations from observations that contain signs with meaningful signifiers and hermeneutic signifieds. Technical images are concepts or ideas visualized with algorithms, data, and computation to become visual, but without referential observed signs to interpret.

9 Technical Images and Live-Coding

To more fully grasp Flusser's line of thought and to position technical images with the context of live-coding, we'll build on Flusser's terminology.

Particles: Atoms or bits of information that follow the Second Law of Thermodynamics by moving towards entropy, meaning that in their natural state, the atoms or bits are in a state of complete randomness.

Technical Image: A mosaic (Image) of particles or bits.

Apparatus: A device that projects the configuration of particles or bits into technical images. For Flusser, to project means to display or visualize; it does not literally mean video projection.

Keys: Buttons that provide the input method for configuring (programming) the particles or bits for the apparatus to display.

Envisioner: The person with the concept who presses the keys to create the program or algorithm. Envisioners configure the particles (bits) to be in states of non-entropy.

To put this together, envisioners program on keys to configure bits on an apparatus that projects (visualizes) those configured bits into a technical image. Stated another way: a person programs a computer to create an image.

Continuing with Flusser's ideas a bit further, the receiver of a technical image can become a critic, and someone who can engage in a dialog with the envisioner regarding the program (2011). This is similar to the description by Rodríguez & Rodríguez who use the terms transmitter (envisioner), receiver (receiver), image (technical image), and communication (discuss) (2015). In this way we can begin to connect Flusser's idea of the critic (receiver) to that of an live-coding audience and Rodríguez & Rodríguez's idea of community.

Our original concept of the Image (the live-coder's projected image) was that it can be a Technical Image (live-coder's projected technical image). According to Flusser, the Image isn't meaningful itself, but the abstract configuration method (program) is. The envisoner and receiver are able to critique and discuss the program's meaning as abstractions without the hermeneutic baggage of the concrete. For example, two people can discuss the abstract, numerical idea of PI without there being a concrete PI to experience.

As an example, consider the live-coding of poetry. If in a technical image I see the letters that Matsuo Basho may have written referring to a frog jumping into a pond, Flusser would say that this is the computation of concepts. There is no real frog jumping into a pond. And besides, which frog? Which pond? How did it jump? To project an image or image sequence of a frog jumping into a pond would then be to represent a more traditional, concrete image. As long as I know what a frog is, and have a good understanding of what it looks like when a frog jumps, the distinction above is obvious. But imagine it's written as code in some language I don't understand.

```
frog.jump() => pond
```

If I don't speak this language (programming or linguistic, which happen to be ChucK and English respectively), then this technical Text-Image truly means nothing to me, supporting Flusser's thesis. Suppose I do know both programming and linguistic languages, then Flusser postulates that I am a technical image critic, and because we know a technical image's meanings is how, not what, I would engage in a dialog of programmatic concepts of the Text with the envisioner. I read the Text as code, or instructions for some action, but not as a graphic element that looks text-ish.

Knowing that technical images have no meaning as they are something out of nothing, we are receiving the visualization that an envisioner has projected out of their programmed bits. In the context of live-coding these programmed bits are visualized as code in the projected Image. If we understand the programming language, what we perceive in the Image is Text, but if not, we perceive the code as a Text-Image.

10 Live-Coded Graphics

Much of this paper has been the concerned with the output of live-coding as textual, and output that is viewable within the Image. The following will concentrate on the live-coding of Graphics as it applies to the Image and through the lenses of Barthes, Foucault, and Flusser.

When applying Barthes's concepts of Text and Graphics to live-coded Graphics we'll consider three cases: Text adjacent Graphic, Text atop Graphic, and Text merged with Graphic. For Barthes when the Text is adjacent to the Graphics they would relate to each other as caption and image. The Text *steals* the connotative message of the Graphic. If the Text is atop the Graphics, the Text would appear to lose its connotative meaning and becomes a Text-Image, unless of course the receiver understands the code of the Text in which case it remains as Text. But it is important to note that Barthes's frame of historical reference is for represented imagery. Let us emphasize here: the re-presention of something that is concrete. In this context Barthes has no concept of algorithmic imagery or of technical images. Speculating, we would think that he retains the Text-Image denotation as before AND the connotation. He has no cultural-historical reference for understanding the graphics, and Barthes may infer that the graphics are once again illustrating the Text. When the Text is merged with the Graphic, Barthes theories appear to not keep up.

When we apply the calligram to describe live-coded graphics we use Foucault's three calligram roles: alphabet augmentation, repetition without rhetoric, and the double cipher. Additionally, the calligram's shape, the output, in the scenario of live-coded Graphics is the Graphic.

First, Text adjacent to Graphics, we previously established in our Python example that Text is augmented alphabet, that the Text and Graphic are tautological, and that the Text is separate from the Graphic. This would give us the same result as the Python example: raveled, raveled, and unraveled.

Second, Text atop Graphic, Text continues as augmented alphabet, tautology holds, Text is on top of the Graphic, meaning that it is visually inside the shape, but more of a composite than an encoding. For the double cipher to work each part must be encoded in the other. The result here is raveled, raveld, and unraveled.

Third, Text merged with Graphic, augmentation remains, tautology remains, the merger of Text and Graphic (Text-Graphic-Image) is more ambiguous. From a visual standpoint, one could argue that Text and Graphic are not really encoding each other; whereas from a conceptual standpoint a Text-Graphic-Image is a symbiotic encoding of Text and Graphic. This is left as an open question, with the thought that while this is possible, live-coding Text merged into Graphic could be incredibly difficult to perform. In this last case we have a result of raveled, raveled, and unclear.

Extending technical images into live-coded Graphics is the easiest. We have demonstrated that the Image is a visualization of the bits, and logically following that is the Text-Image. For Text adjacent to Graphic and Text atop Graphic, there is only the addition of a second visualization method of the bits into the Graphic form. The exact same bits can be visualized as Text and visualized as Graphics. The difference with Text merged with Graphic into a Text-Graphic-Image is that there is a single visualization.

In all of these, it's clear that Flusser's idea of the technical image is most relevant to the live-coder's projected image. Moreover, in the calligram thought experiment earlier, it might be possible to use Flusser's definition of particles to complete the double cipher raveling of Foucault's calligram. In this case, Barthes's concepts become relegated to only traditional images.

11 Annotations

Live-code annotations are bits of Text or Graphic in the Image that provide additional information. For example, highlighting a line of code to show where an error may be, bolded Text to demonstrate which line or block is currently executing, or Text output from a terminal prompt.

Barthes could have relevance here if the annotations take on an iconographic form, which when thought of as a sign could hold some denoted interpretation. For Foucault's calligram, the annotations might be a shape (output), although annotations typically change, we find Foucault stumbling to define these attributes as calligrams are static images. Postulating from the perspective of Flusser, annotations are another form of visualization. The data within the bits is visualized in another textual or graphic form. If the live-coding is of graphics, then the annotation would be the third visualization. First the Text, second the Graphics, third the Annotation, all of which are visualizations of the underlying programmed bits.

12 The Terms

Over the course of this paper, we have been positioning our investigation with respect to the specific contents of the projected image itself. How can we accurately describe what it is we see within the projected frame, and specifically, how can we discern the various layers of visual content? That framework is defined as follows:

Text: The characters, ideograms, or other equivalent methods of programming/written communication. At its most basic the Text would be the live-coder's code.

Graphic: Any visual element that is not the Text but is directly related to the Text. Naturally, the use of text as a graphic element is well within the boundaries of this definition, although that text would be independent of the Text as defined above.

Image: The entirety of the live-coder's projected image. For the purposes of this framework, the Image does not include anything beyond the boundaries of the projection.

Null-Image: The absence of the live-coder's projected screen.

Text-Image: This combination of Text and Image refers to the live-coder's Text as a Graphical abstraction of its intended function. The live-coder's Text becomes a Text-Image when the Text cannot be interpreted, or the viewer chooses not to interpet it.

Graphic-Image: The Graphic-Image includes all Graphic elements within the Image, while the Text is absent.

Text-Graphic-Image: Text-Graphic-Image describes an Image that contains merged Text and Graphics.

Annotation: Informative or assistive data that visually represents some functional aspect.

Text-Annotation: An annotation in Text form.

Graphic-Annotation: An annotation in Graphic form.

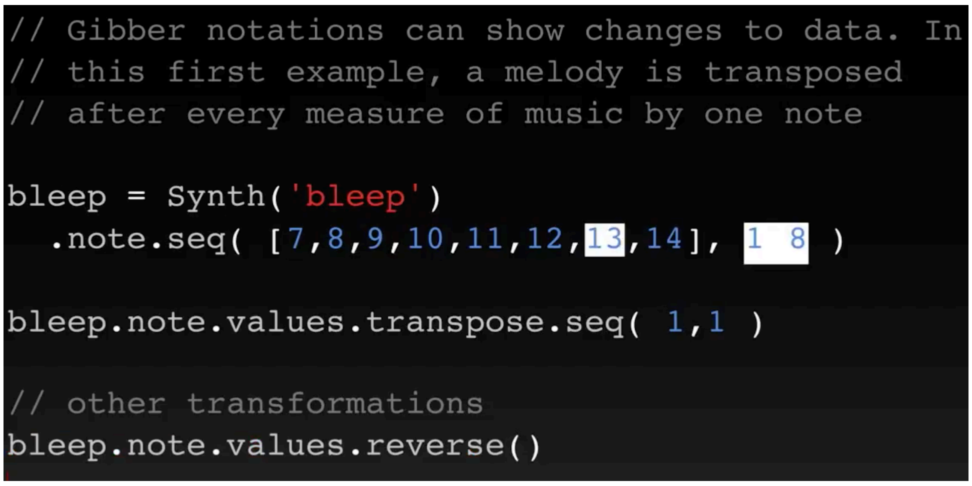
Interface: Any visual element that is not the Text, Annotation, or Graphic as defined above for the functional purpose of user interaction with the apparatus.

Text-Interface: Interface in text form, like that of a drop-down menu.

Graphic-Interface: Interface in a graphic form, like that of a color-chooser.

13 Application of Terms

As a test of this framework we examined several examples of live-coding practices, and considered each as though they were being projected.



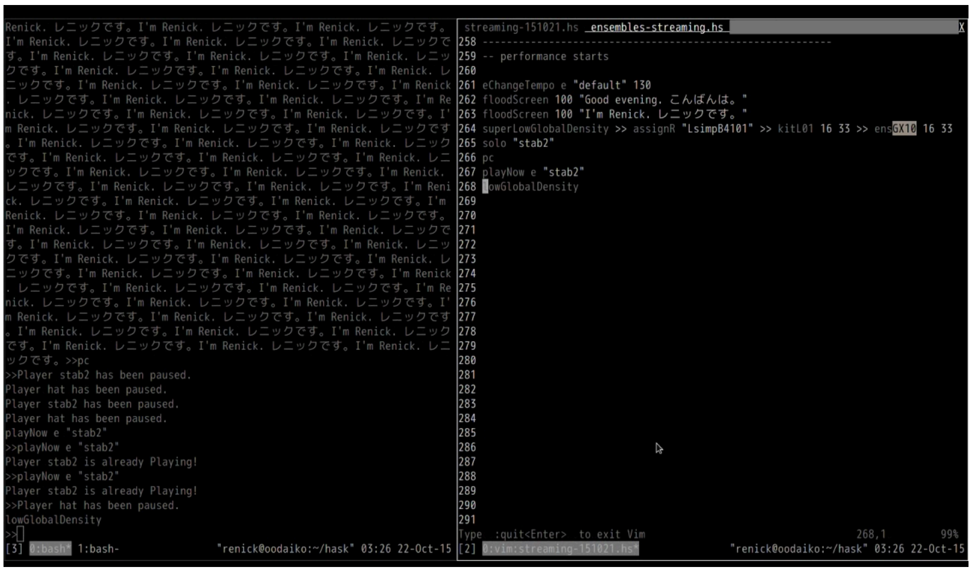
```
// Gibber notations can show changes to data. In
// this first example, a melody is transposed
// after every measure of music by one note

bleep = Synth('bleep')
  .note.seq( [7,8,9,10,11,12,13,14], 1 8 )

bleep.note.values.transpose.seq( 1,1 )

// other transformations
bleep.note.values.reverse()
```

Figure 5: **Screenshot of Charlie Roberts demonstrating Gibber with Javascript in the Gibber IDE**

The screenshot, figure 5[4], of Gibber by Charlie Roberts shows several lines of code, the Text, that produces a rising or descending scale with transposition. If we understand the functionality of the Text, we would describe this as an

[4]https://www.youtube.com/watch?v=iwbmJWw-0z8

Image that contains Text. If not, the Text would be described as a Text-Image. There are no Graphics contained within the Image so many of the other descriptors are not applicable. However, the highlighted "13" and "1 8" represent an annotation. The comments in the Image read "Gibber notations can show changes to data" and these white highlights flash in correspondence to the current pitch and the beginning of each sonic event respectively. Given that this annotation is a white rectangle we would refer to this as Graphic-Annotation, and again, depending on our understanding of the Text, could describe this example of Gibber as an Image that includes Text (or Text-Image if are not familiar with the language) and Graphic-Annotation.

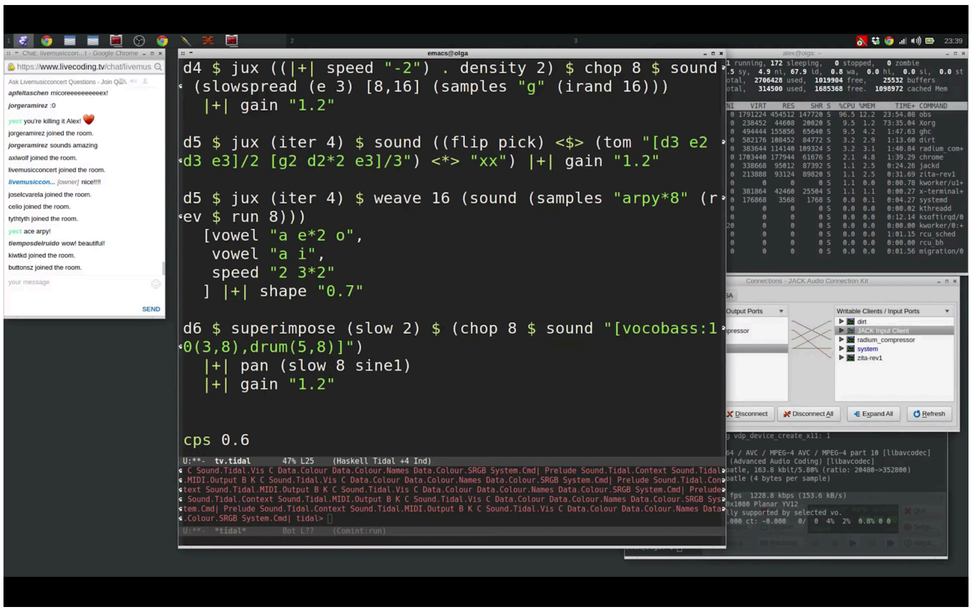


```
Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。
I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックで
す。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニッ
クです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レ
ニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick
. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Re
nick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'
m Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです
。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニック
です。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニ
ックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick.
レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Reni
ck. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm
Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。
I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックで
す。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニッ
クです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レ
ニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick
. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'm Re
nick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです。I'
m Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニックです
。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニック
です。I'm Renick. レニックです。I'm Renick. レニックです。I'm Renick. レニ
ックです。>>pc
>>Player stab2 has been paused.
Player hat has been paused.
Player stab2 has been paused.
Player hat has been paused.
playNow e "stab2"
>>playNow e "stab2"
Player stab2 is already Playing!
>>playNow e "stab2"
Player stab2 is already Playing!
>>Player hat has been paused.
lowGlobalDensity
>>
[3] 0:bash* 1:bash-            "renick@oodaiko:~/hask" 03:26 22-Oct-15
```

```
 streaming-151021.hs  ensembles-streaming.hs                          X
--------------------------------------------------------------
-- performance starts

eChangeTempo e "default" 130
floodScreen 100 "Good evening. こんばんは。"
floodScreen 100 "I'm Renick. レニックです。"
superLowGlobalDensity >> assignR "LsimpB4101" >> kitL01 16 33 >> ensGX10 16 33
solo "stab2"
pc
playNow e "stab2"
lowGlobalDensity

Type  :quit<Enter>  to exit Vim                            268,1         99%
[2] 0:vim:streaming-151021.hs*          "renick@oodaiko:~/hask" 03:26 22-Oct-15
```

Figure 6: **Screenshot of Renick Bell performing a live stream to the Idiotic Code: On Resistant Usership seminar (2015) with Conductive in VIM.**

A different form of annotation can be seen in the example by Renick Bell, figure 6[5]. The Text is confined to the right side of the screen with the left side reserved for printing commands from the Text as well as reporting back the state of the system. Unlike Gibber, this type of annotation would be described as Text-Annotation as the Annotation follows the same language as the Text. This example of the Image contains Text (again, so long as we understand it) and Text-Annotation.



```
emacs@olga
d4 $ jux ((|+| speed "-2") . density 2) $ chop 8 $ sound
 (slowspread (e 3) [8,16] (samples "g" (irand 16)))
  |+| gain "1.2"

d5 $ jux (iter 4) $ sound ((flip pick) <$> (tom "[d3 e2
d3 e3]/2 [g2 d2*2 e3]/3") <*> "xx") |+| gain "1.2"

d5 $ jux (iter 4) $ weave 16 (sound (samples "arpy*8" (r
ev $ run 8)))
  [vowel "a e*2 o",
   vowel "a i",
   speed "2 3*2"
  ] |+| shape "0.7"

d6 $ superimpose (slow 2) $ (chop 8 $ sound "[vocobass:1
0(3,8),drum(5,8)]")
  |+| pan (slow 8 sine1)
  |+| gain "1.2"


cps 0.6
U:**-  tv.tidal       47% L25    (Haskell Tidal +4 Ind)
 C Sound.Tidal.Vis C Data.Colour Data.Colour.Names Data.Colour.SRGB System.Cmd| Prelude Sound.Tidal.Context Sound.Tidal
.MIDI.Output B K C Sound.Tidal.Vis C Data.Colour Data.Colour.Names Data.Colour.SRGB System.Cmd| Prelude Sound.Tidal.Con
text Sound.Tidal.MIDI.Output B K C Sound.Tidal.Vis C Data.Colour Data.Colour.Names Data.Colour.SRGB System.Cmd| Prelude
 Sound.Tidal.Context Sound.Tidal.MIDI.Output B K C Sound.Tidal.Vis C Data.Colour Data.Colour.Names Data.Colour.SRGB Sys
tem.Cmd| Prelude Sound.Tidal.Context Sound.Tidal.MIDI.Output B K C Sound.Tidal.Vis C Data.Colour Data.Colour.Names Data
.Colour.SRGB System.Cmd| tidal>
U:**-  *tidal*        Bot L77    (Comint:run)
```

Figure 7: **Screenshot of Alex McLean performing a live stream improvisation (2015) with TidalCycles in EMACS**

In our next example we look at Alex McLean's TidalCycles. In figure 7[6], the first item we see is the familiar Text occupying center screen, and we will make the assumption that in this case the viewer has an understanding of how TidalCycles

[5]https://www.youtube.com/watch?v=oTMuzM_-_0M
[6]https://www.youtube.com/watch?v=8y_47ExSLRE&t=789s

works. To the right side of the screen are several windows that provide Text-Annotations regarding the consumption of various processes on the CPU, and below the Text a string of Text-Annotations that provide information on the state of TidalCycles. To the left is a chat window that indicates who has logged into the livestream, who has left, and who might have something to say while they are present. This too is a Text-Annotation and the heart emoji is a Graphic-Annotation, but like the Text-Annotations along the right side of the screen, these Text-Annotations are only tangentially related to the live-coding happening in the center. Yet, given that they are included in the Image, they contribute to the Image as a whole. Similarly, the application icons along the top edge of the screen are within the Image, yet their functional independence from the Text renders them Interface-Graphic. This particular Image could be described as containing Text, Text-Annotation, Graphic-Annotation, and Interface-Graphic.

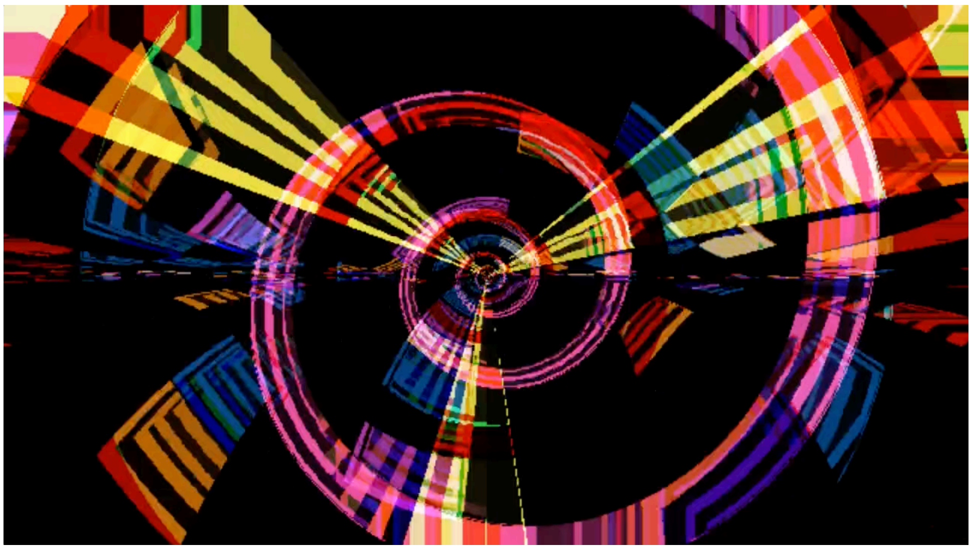
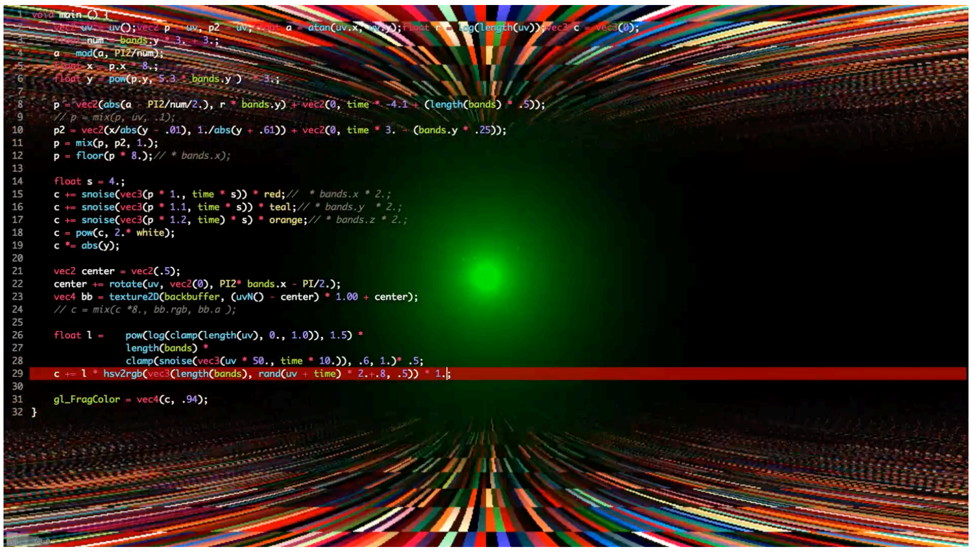


Figure 8: **Screenshot of Ryan Ross Smith and Shawn Lawson performing *EV9D9* (2017) at the GENERATE! Festival with Tidal Cycles and GLSL in The Dark Side IDE**

For our final example we use The Force by Shawn Lawson, who is also a co-author of this paper. The Force uses the OpenGL Fragment Shader as its live-coding language for graphics, and the following screenshot, figure 8[7], is an example of The Force operating in Graphic-only mode, creating an Image that contains a Graphic-Image.



```
void main () {
    vec2 uv = uv();vec2 p = uv, p2 = uv;float a = atan(uv.x, uv.y);float r = log(length(uv));vec3 c = vec3(0);
    float num = bands.y * 3. + 3.;
    a = mod(a, PI2/num);
    float x = p.x * 8.;
    float y = pow(p.y, 5.3 * bands.y ) * 3.;

    p = vec2(abs(a - PI2/num/2.), r * bands.y) + vec2(0, time * -4.1 + (length(bands) * .5));
    // p = mix(p, uv, .1);
    p2 = vec2(x/abs(y - .01), 1./abs(y + .61)) + vec2(0, time * 3. - (bands.y * .25));
    p = mix(p, p2, 1.);
    p = floor(p * 8.);// * bands.x);

    float s = 4.;
    c += snoise(vec3(p * 1., time * s)) * red;//  * bands.x * 2.;
    c += snoise(vec3(p * 1.1, time * s)) * teal;// * bands.y  * 2.;
    c += snoise(vec3(p * 1.2, time) * s) * orange;// * bands.z * 2.;
    c = pow(c, 2.* white);
    c *= abs(y);

    vec2 center = vec2(.5);
    center += rotate(uv, vec2(0), PI2* bands.x - PI/2.);
    vec4 bb = texture2D(backbuffer, (uvN() - center) * 1.00 + center);
    // c = mix(c *8., bb.rgb, bb.a );

    float l =    pow(log(clamp(length(uv), 0., 1.0)), 1.5) *
                 length(bands) *
                 clamp(snoise(vec3(uv * 50., time * 10.)), .6, 1.)* .5;
    c += l * hsv2rgb(vec3(length(bands), rand(uv + time) * 2.+.8, .5)) * 1.;

    gl_FragColor = vec4(c, .94);
}
```

Figure 9: **Screenshot of Mike Hodnick and Shawn Lawson performing a *Mint* (2016) recording with Tidal Cycles and GLSL in The Dark Side IDE**

If the Text were visible in The Force, as in figure 9[8], then the Image would contain Text (to reiterate again, if we don't understand the code then we refer to it as Text-Image), Graphics, Graphic-Annotations for audio input frequency infor-

[7]https://vimeo.com/261648424
[8]https://vimeo.com/192920872

mation and error highlights, and a Text-Annotation of render frame rate. Both audio and frame rate annotations are in the bottom left in light gray.

The examples in this section should not be considered exhaustive. There exist many different live-coding languages and many different live-coders, each with their own individual style. In addition, the authors are less familiar with visual programming languages and these, in particular, could be explored in more depth.

14 Conclusion

Over the course of this paper we attempted to employ traditional methods of image rhetorics through Roland Barthes and a unique method of Text and Image combination with Foucault's calligram to better understand the live-coder's Image. In the end we found Vilém Flusser's idea of the technical image to match most closely to a live-coder's projected image. Through this technical image definition we realized with Carolyn Kane that these images are post-hermeneutic and devoid of the traditional image models of meaning interpretation. To truly grasp the Image, the receivers (audience) must critique with the envisioners their conceptual programming ideas.

By developing this framework based around the Image as the container for anything that may appear on the live-coder's projected screen we hope to develop a better understanding of the distinctions that may exist between visual elements. Most important was the distinction between Graphic and Text elements, and how to consider the Text as a visual element while remaining independent of Graphic elements. It is unclear at this point how well this framework would do under a more rigorous stress test incorporating a broader selection of approaches to live-coding.

References

Barthes, Roland, and Stephen Heath. 1977. *Image, Music, Text.* New York: Hill; Wang.

Burland, Karen, and Alex McLean. 2016. "Understanding Live Coding Events." *International Journal of Performance Arts and Digital Media* 12 (2): 139–51. https://doi.org/10.1080/14794713.2016.1227596.

Cox, Geoff, and Alex McLean. 2013. *Speaking Code: Coding as Aesthetic and Political Expression.* Computer software Studies. Cambridge, Mass.: The MIT Press.

Cox, Geoff, Alex McLean, and Ward Adrian. 2004. In, edited by Olga Goriunova and Alexei Shulgin, 2004 ed, 161–74. Aarhus: Digital Aesthetics Research Centre, University of Aarhus.

Flusser, Vilém. 2011. *Into the Universe of Technical Images.* Electronic Mediations. Minneapolis: University of Minnesota Press.

Foucault, Michel. 1983. *This Is Not a Pipe.* Berkeley: University of California Press.

Kane, Carolyn L. 2014. *Chromatic Algorithms: Synthetic Color, Computer Art, and Aesthetics After Code.* Chicago, IL: The University of Chicago Press.

Machuca, Mauro Herrera, Jaime Alonso Lobato Cardoso, José Alberto Torres Cerro, and Fernando Javier Lomelí Bravo. 2016. "Live Coding for All: Three Creative Approaches to Live Coding for Non-Programmers." *International Journal of Performance Arts and Digital Media* 12 (2): 187–94. https://doi.org/10.1080/14794713.2016.1227598.

Magnusson, Thor. 2014. "Algorithms as Scores: Coding Live Music." In *NIME'14 Proceedings.* New Interfaces for Musical Expression.

McLean, Alex, Dave Griffiths, Nick Collins, and Geraint Wiggins. 2010. "Visualisation of Live Code." In *Proceedings of the 2010 International Conference on Electronic Visualisation and the Arts*, 26–30. EVA'10. Swindon, UK: BCS Learning & Development Ltd. http://dl.acm.org/citation.cfm?id=2227180.2227185.

Patterson, Zabet. 2015. *Peripheral Vision: Bell Labs, the S-c 4020, and the Origins of Computer Art.* Platform Studies. MIT Press.

Roberts, Charles. 2016. "Code as Information and Code as Spectacle." *International Journal of Performance Arts and Digital Media* 12 (2): 201–6. https://doi.org/10.1080/14794713.2016.1227602.

Rodríguez, Jessica, and Rolando Rodríguez. 2015. "LiveCoding Readings. Algorithms Viewed as Text/Image." In *Proceedings of the 21st International Symposium on Electronic Art - Isea2015: Disruption.* ISEA 2015. Vancouver, BC, Canada: Simon Fraiser University.

Sorensen, Andrew, Ben Swift, and Alistair Riddell. 2014. "The Many Meanings of Live Coding." *Computer Music Journal* 38 (1): 65–76. https://doi.org/10.1162/comj_a_00230.

Ward, Adrian, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Collins Nick, and Amy Alexander. 2004. In, edited by Olga Goriunova and Alexei Shulgin, 2004 ed, 242–61. Aarhus: Digital Aesthetics Research Centre, University of Aarhus.

zmölnig, IOhannes m. 2016. "Audience Perception of Code." *International Journal of Performance Arts and Digital Media* 12 (2): 207–12. https://doi.org/10.1080/14794713.2016.1227604.