Visual Tactics Toward an Ethical Debugging

Catherine Griffiths

Abstract

To advance design research into a critical study of artificially intelligent algorithms, strategies from the fields of critical code studies and data visualisation are combined to propose a methodology for computational visualisation. By opening the algorithmic black box to think through the meaning created by structure and process, computational visualisation seeks to elucidate the complexity and obfuscation at the heart of artificial intelligence systems. There are rising ethical dilemmas that are a consequence of the use of machine learning algorithms in socially sensitive spaces, such as in determining criminal sentencing, job performance, or access to welfare. This is in part due to the lack of a theoretical framework to understand how and why decisions are made at the algorithmic level. The ethical implications are becoming more severe as such algorithmic decision-making is being given higher authority while there is a simultaneous blind spot in where and how biases arise. Computational visualisation, as a method, explores how contemporary visual design tactics including generative design and interaction design, can intersect with a critical exegesis of algorithms to challenge the black box and obfuscation of machine learning and work toward an ethical debugging of biases in such systems.

Critical Code

From a design research perspective, the move to look beyond the omnipotence of software interfaces began in the early 2000s, with Lev Manovich's re-conception of software as the meta-medium and his call to analyse its underlying logic, structure, and processes rather than its observable effects. (2002, 2014) The field of Software Studies was conceived as a type of reflexive thinking around computation and its cultural and societal consequences, questioning the assumed invisibility and neutrality of software by demonstrating its inbuilt political, social, and cultural biases. Noah Wardrip-Fruin picked up this convergence around software in his book *Expressive Processing*, in which he thinks through the techno-cultural transition from having diverse machines built for specific purposes, to the nature of the computer as simulating the functions of many different types of machines,

DOI 10.14361/dcs-2018-0113

including machines that we have never seen before. (2009) To extend this thinking, we could consider the algorithm as a further level of convergence; the algorithm has now become the medium, technology, site of study, form of expression, place of scholarly critique, and ethical tension. In this age of far-reaching data collection, analytics, and machine learning, a consolidation is taking place around a cluster of specific algorithms and techniques that simulate and underlie many of the new power structures driving culture. Marshall McLuhan's argument that the medium is the message continues to be relevant as a recursive trend toward the encapsulation of culture and power by algorithms. (1964) Critical Code Studies is a new field of inquiry that has emerged from Software Studies and the Digital Humanities to focus specifically on source code as a neglected but omnipresent cultural text that should be available for humanistic interpretation. Through close readings of code's structure and syntax, code can be resituated within a social and political value system, in parallel with its executable function, but understood beyond the tradition of mathematical neutrality and technical mystification.

There is an assumption that the only interpreter of code is the computer, but this omits to recognise that code is often written by teams of people in a collaborative environment. Code is often revised and forked by multiple people who come to it at different times and from different perspectives, as well as the hacker who also arrives at code with their particular agenda. Code is not written solely to be executed by a computer, but this emphasis on operability neglects the meaning code bears in culture, its social history, and political context, and diminishes the value in a broad range of analytical tools that have been developed. Critical Code Studies is a call to read and explicate code with the same approach that one might explicate a work of literature, to apply critical hermeneutics to code in a social context. "CCS holds that lines of code are not value-neutral and can be analysed using the theoretical approaches applied to other semiotic systems in addition to particular interpretive methods developed particularly for the discussions of programs." (Marino 2006) While a lack of programming literacy is a limitation to this method of study, it should be noted that there are multiple ways into a critical interpretation of code as text and its production of meaning, including for those without a strong programming background. As Marino states: "People like to project humanity onto the computer, but is it possible that with regard to coding we do just the opposite and strip the code of its human significance, imagining that it is a sign system within which the extensive analyses of semiotic systems and signification, connotation, and denotation do not apply?" (2014) An array of tactics to derive insight from source code include:

- Reading the natural language comments that are written within the source code to remind the programmer how different parts of the program work or to support team development (program should be understood as the coded instructions to automate a task);
- reading programmer-assigned arbitrary variable or function names;

- reading the source code's documentation;
- modification of code as a textual intervention to compare how a change in syntax can execute differently;
- thinking through the implications of how a program would execute differently if ported to a different language or hardware to consider the nature of translation, interpretation, and adaptation when applied to code (several *weird* programming languages have been developed in order to test, provoke, or comment on the nature of programming languages);
- an analysis of computational structures in the context of a program in a given culture;
- an analysis of paradigmatic choices made in the construction of a program;
- an interpretation of methods chosen over others and their connotations;
- a consideration of paratextual features;
- contextualising the history of the program and its author;
- consideration of the programming language used;
- contextualising the funding source for the research and development;
- the use of an algorithm as a design tool by emphasising its visual and aesthetic dimension, to explore alternative patterns as an exploratory interventionist exercise;

all of which all shape meaning, and are available to interpretation. (Marino 2014, Monfort 2013) This is an incipient and still unfolding research territory, however if we want to break with the essentialist and mystified nature of code before we sink below it, we need to plot a line that sensitively brokers the need to address the technicality of code, which can at times feel like an act of "fetishization" (Chun 2011), with the need to conduct a humanistic interpretation, which can at times border on the metaphorical and feel disconnected from its material implementation. At the Humanities and Critical Code Studies lab at the University of Southern California, interdisciplinary teams have conducted critical exegeses of code from the perspective of gender, race, and creativity.

From Data Visualisation to Computational Visualisation

Data visualization, information design, info graphics, are terms that describe "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition". (Card 1999) The field has burgeoned over the past decade, in tandem with the exponential rise of big datasets. In this field, data is the input to the system, the raw material, and the output is an abstracted, representational, point of view of the data in the form of a visualisation. Data visualisation emerged with noble intentions. It attempts to address the complexity at the heart of many social and political issues, and technical studies through the vast repositories of data that have become available. It enables us to approach dense and voluminous content, parse its obfuscated nature, make use of it, and form perspectives on it, enabling another type of cognition: "revealing patterns and relationships not known or not so easily deduced without the aid of the visual representation of information". (Meirelles 2013: 11) Where it initially offered entry to a previously inaccessible or intangible scientific space, and through that gave us access to a broader scale of thinking about messy problems, data visualisation cannot escape its subjective status that is as much about the realm of the visual, and how aesthetics can simultaneously inform, deceive, persuade, and empower. In *Design For Information*, Isabel Meirelles refers to data visualisations as "cognitive artifacts" (2013: 13) for the way in which they support our understanding of information. The term artefact also reinforces an idea of data visualisation as an output or consequence of a process that has taken place in the past. However, it is the process of interpretation, which in the era of big data is algorithmic, that remains black-boxed, and this is where a methodology to visualise algorithms and computational process, takes its departure.

Computational Visualisation is a proposal for a new methodology that combines a close critical reading of source code with tactics of visualisation, to develop a critical inquiry into algorithms as design research. Computational visualisation can be a method of study, and a means of access for a broader audience to engage with algorithms, which would otherwise require a technical knowledge of programming. A key component of this method is its focus on process, both temporally and spatially, in which data is parsed, forked, and on which decisions are executed. It is about thinking through and visualising how the computational process works in real time, to expose or interpret a cause or pattern or resulting artefact. In data visualisation, a designer or analyst begins with a static data set but does not question how the data came to exist, or how the algorithm that parses and mathematically restructures the data functions or arrives at its decisions. Computational visualisation seeks to understand how the algorithm executes and why it produces the results it does, whether this is a conventional sorting algorithm, or a computer vision algorithm with significant social and political implications. In this way, it is possible to access and visualise the processes that underlie the computational systems that increasingly drive key functions in our society. Temporally, data is fixed and retrospective; it is pertinent to think of data visualisation as addressing something that has taken place in the past, in the sense that a data set is an account of a situation that has now ended, at least in terms of the data it contains. Computational visualisation, on the other hand, can be both in process and a projection to the future, in the sense that it is identifying a process that is currently in use to understand how it unfolds. When we visualize an algorithmic process, we can see the decision model that it creates, and it becomes a weapon of computational literacy to help decide which model is best to use. In data visualisation, there is no data model or decision model, and therefore there is no alternative to the understanding of how we arrived at a particular visual solution. Computational visualisation is about identifying a model, then visualising it to share and propagate that knowledge, and cultivate greater computational literacy.

Precedents of Visual Tactics

Computational visualisation can look to practices across interaction design, generative design, and programming games, to adapt and extend visual tactics for the investigation and communication of algorithms. In *Expressive Processing*, Wardrip-Fruin thinks through how computational process is becoming an important means of authorial expression and critique in areas of game simulation, interactive design, and AI, because "the shapes of computational processes are distinctive – and connected to histories, economics, and schools of thought [...] they can be seen as 'operationalized' models of these subjects, expressing a position through their shapes and workings". (2009: 4) Rather than prioritising the creation of content and its representational status, we can think through the design of rule sets and system behaviours, as tools for simulating new processes and expressions. The data visualisation specialist and co-developer of the D3.js library, Mike Bostock, has explored the nature of visualising algorithmic process in his work. He writes: "To visualize an algorithm, we don't merely fit data to a chart; there is no primary dataset. Instead there are logical rules that describe behaviour." (2014) To visualise an algorithm it is the rule sets that need to be identified and given graphic form, and the computational process that contains meaning, rather than a resulting pattern that is gleaned from data. By looking at the structure of the process and the rule sets, we are studying how the system is composed, and from here we can more easily simulate that system to execute with slightly different rules, or with a different structure, and think through what alternative outcomes are possible, or how specific results can be manipulated. This is not possible when only visualising patterns in data. In Bostock's study of a sampling algorithm, which is used to simulate human vision and the biological process at work in the human eye, he develops a reflexive insight into the field of CGI. Bostock works through the concept of scientific randomness and visualises how the random sampling process works, leading to a comparison with other models of generating randomness, to contrast how these other computational processes affect visualisation. From uniform sampling, to Mitchell's best-candidate algorithm, to Bridson's algorithm for Poisson-disc sampling, in each we can see the computational flaws of the algorithm, such as oversampling or undersampling, or the speed it can execute in relation to the outcome, as well as the effect it has on image generation.

The interface designer, Bret Victor, has also explored how we can rethink computational process as it pertains to the act of programming. Victor lays out several ways in which the real-time connection between the process of writing code is separated from the creative outcome, which is necessary to foster a greater sense of discovery and more powerful ways of thinking. He points to the temporal separation created by compiling code, and the visual separation between each part of the code's syntax and its outcome. Also, spatially, there is a separation between pixel representation and syntax, and structurally there is a disconnection between the present state of a system or variable and a map of its past and future trajectories, which could provide an immediate sense of its possibility space. He proposes to create a stronger feedback loop between code, comprehension, and creativity, by revealing the behavioural data within the computational process, revealing simultaneous comparative computations, controlling time to visualise flow, and enabling the *shape* of an algorithm to be understood beyond lines of code. (Victor 2012, 2012)

Within the genre of programming games insights can be gleaned into visualising computational phenomena. SpaceChem is a computer game by Zachtronics that explores systems thinking and computational concepts. SpaceChem is a puzzle game contextualized around a chemical manufacturing company, in which a player performs the role of a Chemical Engineer who must configure atoms into molecular combinations which are processed by chemical reactors to produce a final product. The interactive process of building chemical systems engages a similar problem-solving process as writing an algorithm, and in the game, the visual solution that the player determines could be likened to a graphical representation of an algorithm. SpaceChem utilises a temporal scale to switch between slow and emergent processing speeds. The player can both view computations at a one-to-one scale, which provides the ability to process computation at a human scale of comprehension, including visually identifying where bugs in the system arise, and also view computational processing at a very fast scale, leading to an emergent understanding of the process. Coloured tracks visualise computational loops that interact, pass data, and synchronize over many executions. Simple discrete looping systems are built first, which are then combined with other systems to emulate the structure of encapsulation and objects in programming languages. It is also a spatial map that visualises state changes, conditional statements, and modifications in process. SpaceChem is itself a visual programming language, with loops, conditionals, and functions that can be combined to form a set of executable instructions in which players build abstract chemical systems. Through gameplay, the player builds a visual algorithm to solve a puzzle. Its open-ended problem-solving nature simulates the practice of writing code and constructing algorithms.

In referencing visual design tactics from other disciplines, including generative design, interaction design, and programming games, possibilities for a field of Computational Visualisation emerge. Ideas for how to slow down computational time, reverse-engineer rules, visually cross-reference specific lines of source code with their isolated moments of execution, visualise bugs, all become tools for critically engaging with algorithms as design research. Looking to machine learning algorithms, which make up some of the more obfuscated systems, such visual tactics can be built into a computational visualisation that explores algorithmic composition, decision-making, and mistakes.

Making A Mind Design Research

Making A Mind is a design research project that aims to develop an application of computational visualisation as a tool for the ethical debugging of algorithms operating in socially sensitive spaces. A decision tree classifier, is one type of machine learning algorithm, which was chosen to initiate a study of machine learning processes, because of its graphical nature and possibility to reverse-engineer individual decisions. The decision tree classifier was generated in Python using the scikit-learn library, and rebuilt from a text file into a 3D interactive and animated simulation in Unity, a game engine. At the time of development, the other simple visualisation tool found for this process was the graphviz Python library, which can construct a diagram of the classifier that is useful as an overview, but functions more as a snapshot of the structural logic and lacks any temporal insight.

The prototype that is currently in development presents a spatial map of the algorithm's data structure that uses generative design techniques of circle packing coupled with spring physics to automatically and organically generate different topologies of classifiers from self-organising properties, which can be physically moved and reconfigured through the interaction of the user. The prototype can also visualise both the continuous flow of a large data set through the algorithm and can zoom in on each data point, whose path through the algorithm can be reverse-engineered. Points of forking and segmentation of data, along with final decisions, are visualised. If one thinks of the visual classifier as the possibility space of the algorithm, popular paths taken through the algorithm can be compared with paths less taken. The data points are deployed using steering behaviours, to follow the classifier's sequential logic gates, allowing them to gradually navigate the system and allowing the user to identify how the data is deconstructed and classified, and which features are more susceptible to a particular classification.

A critical component of the simulation is to sync computational time with human perception in order to reveal the flow of data through the algorithm, individual points of decision-making, the visualisation of errors, and potentially the accumulation of bias. This is achieved by slowing down every process to a simulated trajectory between an initial condition and the final location achieved after classification. In computational time this process happens simultaneously, and it is easy to overlook how the algorithm can make incorrect decisions at individual nodes. Time, in this case, should be mapped to a variable, enabling acceleration, deceleration, or even reversal of the process like an animation timeline, allowing for the analyst to be able to explore the software with greater ease. Scrutiny can also happen through magnification, where the user can zoom in space and time, revealing details obscured in an otherwise compressed and inaccessible format.

Beginning with simple data sets to test the visualisation capacity of the prototype, the project has slowly progressed to experiment with more complex but synthetic data sets, in which the system is challenged by the number of features in relation to the number of classes, and to accept both continuous and discrete data sets. Uses of synthetic data tactically neutralizes the meaning of data to instead focus on the data structure and shapes of different classifier algorithms based on different data sets (features versus targets versus discrete/continuous sets). Uses of socially meaningful data focus on where in the algorithm, spatially, biases or errors might occur, leading toward an ethical debugging. The reason behind working with synthetic data lies the question of whether it is possible to detect biases or incongruences by looking purely at the data structure in isolation from the data. Whilst it is known that many ethical problems can be traced back to discrimination and manipulation in the original data, the question here is to understand whether the data structure or algorithmic process can also reveal discrimination, either alone or by means of augmenting a latent bias in the data set. The more recent stage of the research moves to working with real-world data sets, specifically data that is charged with known discrimination or socially sensitive potential. While the prototype cannot currently pinpoint bias, it works toward developing the discourse around what is the scale and landscape of action in regard to comprehending algorithms and ethics. Does bias lie solely in the data, as frequently stated, or can it also lie in the structure of the classifier, and perhaps in the process that couples those together. These are the questions that guide this design research project as it builds toward more complex machine learning algorithms, those which cannot be reverse-engineered, and yet have greater consequences for the ethics in algorithms debate.

Making A Mind is a design research prototype for visualising a machine-learning algorithm, and can currently visualise errors, which is when the algorithm has misclassified data. More broadly, it provides a system for interrogating the inner dynamics of an algorithm, which can otherwise only be accessed through reading source code, and to frame further enquiry. The ethical dilemmas that are arising from the use of machine learning algorithms include the likelihood of them generating mistakes and of augmenting biases hidden in data. An investigation by ProPublica showed how such systems can encode racial bias when used in criminal sentencing. (Angwin 2016) Cathy O'Neil's work points out how algorithms used in the workplace, including in job appraisal and in recruitment, far from being neutral mathematical arbiters, can fuel inequality and poverty, due to them being opaque, untested, and beyond redress. (2016) In the context of this design research, ethics is understood in terms of current debates around the application of machine learning in areas in which discrimination can be encoded, poverty exacerbated, and inequality engineered. Whilst statistical models have been used in decision-making previously, machine learning algorithms, defined by the capacity to generate and modify their own decision model autonomously, introduce a significant level of uncertainty and opacity over how and why decisions are made, inhibiting appeal when ethical questions arise. (Mittelstadt 2016)

In the ethics in algorithms debate, it is common to understand algorithms, not only as mathematical constructs programmed to execute on a computer, but as human-algorithmic agent implementations in socio-technical systems, where responsibility is shared across the network. According to this position, it is problematic to fixate on source code as holding the key to unravelling ethical problems. Looking inside the black box does not mean that an algorithm becomes transparent and therefore accountable, as transparency assumes that what is inside is discernible and that the audience is able to comprehend it. (Ananny 2016) However, in this design research prototype, a preoccupation with code is advocated in favour of enabling greater computational literacy for how algorithms work, especially during a time when they are becoming more opaque whilst simultaneously being given greater authority and autonomy. Knowledge can be gained by examining the composition of an algorithm, and to neglect this is to overlook the way algorithms frame problems and contain singular important statements. A study by Sandvig et al on whether an algorithm itself can be racist, describes how an algorithm can be written in multiple ways to achieve the same result, encoding normatively positive behaviour, defining variable ranges within which to identify skin colour, or categorizing race when that would not be an acceptable question, perhaps even an illegal question, in the context in which the algorithm is being used. (2016) It follows that if the source code is neglected, this type of unethical behaviour cannot be determined. Reading the code can be a useful strategy to understand algorithms and how ethics can be encoded at the level of syntax. The Making A Mind prototype, as a visual tool, provides a way to re-engage with decision-making again, in conjunction with machine learning algorithms. The research is ongoing and aims to identify ethical biases, however independently from its ability to generate ethically meaningful results, the tool has been built to contribute to the democratic and political imperative of providing access to and comprehension of algorithms that are increasingly afforded greater roles in governance and society.

References

Ananny, Mike. and Crawford, Kate (2016): "Seeing without knowing: Limitations of the transparency ideal and its application to algorithmic accountability". In: *New Media and Society*, DOI: 10.1177/1461444816676645

Angwin, Julia, Larson, Jeff, Mattu, Surya, and Kirchner, Lauren (2016): Machine Bias. In: *ProPublica*, May 23rd 2016, https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Bostock, Mike (2014): "Visualizing Algorithms". June 26th 2014, https://bost.ocks.org/mike/algorithms/

Card, Stuart, Mackinlay, Jock. D, and Shneiderman, Ben (1999): *Information Visualization: Using Vision To Think*. Morgan Kaufmann

Chun, Wendy Hui Kyong (2011): *Programmed Visions: Software and Memory*. MIT Press

Manovich, Lev (2002): *The Language Of New Media*. MIT Press

Manovich, Lev. (2013): *Software Takes Command*. Bloomsbury

Manovich, Lev. (2014): "Software Is The Message". In: Journal of Visual Culture, DOI: 10.1177/1470412913509459

Marino, Mark. (2006): "Critical Code Studies". In: *Electronic Review of Books* http://www.electronicbookreview.com/thread/electropoetics/codology

Marino, Mark. (2014): "Field Report on Critical Code Studies". In: *Computational Culture* http://computationalculture.net/field-report-for-critical-code-studies-2014%E2%80%A8/

McLuhan, Marshall. (1964): *Understanding Media: The Extensions of Man*. MIT Press

Meirelles, Isabel. (2013): *Design For Information*. Rockport Publishers

Mittelstadt, Brent, Daniel, Allo, Patrick, Taddeo, Mariarosario, Wachter, Sandra, and Floridi, Luciana. (2016): "The Ethics of Algorithms: Mapping The Debate". In: *Big Data & Society*, DOI: 10.1177/2053951716679679

Montfort, Nick, Baudoin, Patsy, Bell, John, Bogost, Ian, Douglass, Jeremy, Marino, Mark. C, Mateas, Michael, Reas, Casey, Sample, Mark, and Vawter, Noah. (2013): *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. MIT Press

O'Neil, Cathy. (2016): *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Press

Sandvig, Christian, Hamiton, Kevin, Karahalios, Karrie, and Langbort, Cedric. (2016): "When the Algorithm Itself Is a Racist: Diagnosing Ethical Harm in the Basic Components of Software". In: *International Journal of Communication*, http://ijoc.org/index.php/ijoc/article/view/6182

Victor, Bret. "Inventing on Principle". Video recording. Canadian University Software Engineering Conference (CUSEC). 2012, http://worrydream.com/#!/InventingOnPrinciple

Victor, Bret "Learnable Programming", September 2012, http://worrydream.com/#!/LearnableProgramming

Wardrip-Fruin, Noah. (2009): *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. MIT Press