

Analysing Digital Maps Online: A Reverse Engineering Approach

Florian Ledermann

Technical University of Vienna, Austria

Abstract

The age of 'Big Data' and rapid online publishing brings with it a new need for verifying and understanding digital media. Cartographers are interested in the analysis of maps published online with regards specifically both to their adherence to established cartographic techniques and to identifying innovative approaches to geographic visualization. But traditional manual cartographic analysis methods, based solely on inspecting a map as an image, are limited; they can neither keep up with the volume of new maps coming online every day, nor answer all questions about the process of creation of the map. This paper proposes a novel approach to analysing maps which takes into account the source code of the cartographic program producing the map image. It is argued that such a method can surpass manual analysis in both its potential to facilitate semi- or fully automated processes as well as the depth of insight that the method is able to give into the complete socio-technical assemblage of a digital map artefact.

Keywords:

digital cartography, web maps, reverse engineering, transformations, deconstruction

1 Introduction

The advent of the web as a distributed publishing platform and the even greater ease with which information can be spread via Web 2.0 and social media have raised unique challenges for researchers and critical observers of media. The '4 V's' of Big Data (Hitzler & Janowicz, 2013) sum up the challenges faced by anyone who wants to engage with information published on social media, or on the web in general: the *volume* and *velocity* of the stream of new messages overwhelm our capacity to engage with their content using traditional techniques for analysing individual media artefacts; the *variety* of content makes it hard to establish a single analytical (or even technical) framework; and mixed *veracity* prevents the naïve application of content summarization methods to obtain a reliable overview of trends and new developments: there is increased awareness of the questionable trustworthiness of unknown actors in the light of the 'fake news' debate, and of the potential for paid actors and bot networks to skew opinions online.

As cartographers, we are mainly interested in the ever-increasing number of maps and other depictions of geographic phenomena posted online. Scrutinizing maps for their adherence to cartographic best practices, cognitive feasibility and the soundness of the depictions of the underlying phenomena has long been an important part of the work of cartographers (Monmonier, 1996). So, too, has looking out for new techniques or noteworthy examples of cartographic production. Typically, cartographers would examine a given map by asking questions relating it to an established body of cartographic knowledge: What projection is used in this map? How are the colours and other visual variables of map symbols assigned? How is the classification of phenomena established? etc. But the developments sketched out above have not left geographic media unaffected. Arguably, more maps than ever are shared online today, and many of those are produced by map makers who do not have a traditional education in cartography, but come, as ‘neocartographers’, from a background as designers, coders or journalists (Faby & Koch, 2010; Cartwright, 2012). With new channels for sharing and distribution, maps can become ‘viral’, potentially reaching massive audiences and triggering the creation of derived maps as ‘memes’ (Robinson, 2018). Furthermore, commercial actors from outside academic and standards-based cartography shape to a large extent our expectations of how maps look and work online today, without necessarily engaging in the open debate and critical reflection on techniques and practices.

In the wider field of digital humanities, Alier and Casany have discussed the challenges that the exponential growth of technology poses for researchers (Alier & Casany, 2017). They see potential in leveraging the expanding power of technology for research, but in the face of limited research resources and limited knowledge on the one hand, and an exponential output of online media on the other, severe challenges remain. In the field of cartography, researchers have developed strategies for ‘keeping pace with evolving web mapping technologies’ (Roth, Donohue, Sack, Wallace, & Buckingham, 2014), but Roth later argues that teaching fundamental concepts is of higher importance in a geography curriculum than focusing on the details of individual technologies (Roth, 2016) – a reasonable position, but the questions remain of how to engage critically with digital media without detailed technical understanding, and what the ‘fundamental concepts’ of a continuously evolving medium even are.

The goal of the ongoing research presented here is to find novel, technology-supported methods for non-engineering researchers to critically analyse digital maps. To develop such methods, another major trend in digital media is taken into account – one that has changed the way in which maps are delivered online: the interaction capabilities of modern web browsers and the need to adapt to different devices and contexts by means of ‘responsive design’ (Marcotte, 2010; Griffin, Robinson, & Roth, 2017) require that many maps are not published online as static ‘flat’ images but instead as interactive experiences, which contain program code to adapt, animate or provide the user with means of interaction. The central research question of this project is: can we use the digital code of online maps to gain further insight into their inner working and creation process, possibly in an automated or semi-automated fashion, in order to keep up with the volume of new maps posted online and the exponential growth of technology?

The idea that program code is a form of social media, with its own social platforms like GitHub, glitch.com or blocks.org for sharing, commenting and ‘remixing’ code, is not new

(Dabbish, Stuart, Tsay, & Herbsleb, 2012). Analysing code as a medium by using social sciences methods such as content analysis or close reading has been practised within critical code studies and software studies for some time (Mackenzie, 2006; Fuller, 2008). However, as cartographers we are not interested in program code as a textual medium per se; an analysis of the program code of maps should seek to further the understanding of the spatial and cartographic transformations applied, and new techniques are needed that can help with such an analysis. Such a cartographic analysis of program code would therefore tie in with a wider interest in how algorithms are used in our society to make decisions and to filter and transform data (Roberge & Seyfert, 2017), and with questions of accountability, quality and the possibility for critique of such automated processes. In an even wider sense, we would hope to contribute to an understanding of what exactly the underlying principles of digital maps are, how we can engage critically with them, and what their role in a wider ‘culture of the digital’ (Stalder, 2016) may be.

To elaborate, and hopefully partially answer the questions raised above, this paper will proceed as follows: a conceptualization of digital maps as coded transformations from data to visual output will be sketched out in the next section, to lay the groundwork of a shared understanding of digital online cartography. In Section 3, the possibility of automatic ‘reverse engineering’ of such coded transformations will be presented, and in the final section some conclusions for future research will be drawn from the findings, including a conceptualization of digital maps as ‘deep images’ that reveal the transformations of data for each of their graphical elements.

2 Understanding digital cartography – from data through code to pixels (and back again)

Cartographers have always asked questions about the quality and origin of maps and their adherence to cartographic best practices. For historic maps, an educated cartographer can estimate whether a map was printed earlier or later in a print run (Campbell, 1989), or can use software tools to assess its cartographic projection and spatial distortion (Jenny & Hurni, 2011). However, such methods based solely on analysing the image of the map require the attention of an expert to each artefact to be analysed, and are limited in fundamental ways. For example, it is in principle not possible to fully assess from a map image whether and which information has been omitted from the map (because there is nothing in the image to analyse). Nor, without access to the map’s creators or additional artefacts that reveal information about the creation process, is it possible to gain insight into the rationale behind some of the cartographic decisions reflected in the map.

As has been pointed out above, for a subset of maps posted online – those which come in the form of program code that is run in the user’s browser to generate the map – we can gain access to such an artefact: the map’s source code. While it seems obvious that the code of the map says *something* about the map, scholars have struggled to find ways to make sense of that information. Kitchin (2017) sums up the debate as code being a ‘big ball of mud’ (a phrase attributed to Foote and Yoder (2000)) and concludes: ‘Even those that have produced it can find it very difficult to unpack its algorithms and routines; those unfamiliar with its

development can often find that the ball of mud remains just that' (Kitchin, 2017, pp. 22–23). How, then, can we proceed to make sense of the 'ball of mud' to analyse spatial and cartographic phenomena?

Galloway (2010) points out that there are three levels of any software, which are connected but do not directly mirror each other. At the level of the software's author is the source code used to write the program; at the level of the machine, the executed code performs calculations and transformations on data; at the user's level, there is the expression of the software on the screen, experienced as pixels, graphics and other elements of a user interface. Analysing source code (the first level) with the goal of learning about its visual output (the map, on the third level) only makes sense if the expressions found in the code can be related to the transformations and calculations (on the second level) producing the output.

Almost 40 years ago, Tobler (1979) established the concept of *transformations* as a key paradigm of cartography, covering a wide range of aspects that one may be interested in in cartographic analysis: cartographic projections, classification of continuous values into distinct categories, colours assigned to the elements on the map from data values, etc. For any digital map, the transformation of raw geographic data (input) to pixels on the screen (output) is performed by the computer and must therefore be expressed in the source code of the software components involved in that process. If an observer has access to that code, she could attempt to read it and thereby reconstruct the transformation processes at work in the map.

Such an approach, based on the 'close reading' of code, has been presented by Ledermann (2016), with the conclusion that it is possible to identify well-known cartographic concepts such as projections or the assignment of visual variables in the source code of maps written for different mapping APIs, but the amount of work needed to analyse even simple examples makes such a method prohibitively labour intensive. One of the difficulties found by this study is that even for small programs dedicated to demonstrating cartographic techniques, the proportion of code that actually relates to cartographic transformations and their visual output is small. So instead of trying to analyse the code of a cartographic program in its totality, an appropriately educated observer could try to locate just the essential portions of the code relevant for cartographic analysis.

Since the map is the starting point for such an analysis, one strategy would be to start with the map image and *work backwards* through the code, at every step asking the question 'Where is the information needed at this point coming from?' and continuing the analysis at these locations. Digital maps on a screen are composed of pixels, so any pixel on the screen can be the starting point for such an investigation. The pixel can be associated with an output primitive, drawn on the screen by the browser and operating system. The shape and position of the output primitive is determined by parameters, which we can locate in the code representing the output primitive. Each parameter, in turn, can be traced back to the location in the code where it is defined by a calculation or constant. Each calculation will originate either in further calculations that can be traced back recursively through the code, or, eventually, to its origin in constant values defined in the code or loaded from external data files.

Figure 1 gives an example of such an analysis of a real-world example running in a web browser (the tutorial ‘Let’s Make a Map’ by Mike Bostock (2012)). The chosen pixel is part of a text label, represented by an SVG `text` element. The attributes defining its position and appearance are `class`, `transform`, `x`, `dy`, `style` and the element’s `text content`. We can trace each of those attributes back through the code – in the example, the value of the `x` parameter (determining the text’s position on the screen) is traced back to a decision made in the code: print all place names west of the meridian of 1° latitude to the left of the place it refers to, and position all place names east of that meridian to the right of the place – a primitive label placement algorithm that is sometimes applied for simple maps. The latitude/longitude coordinates of the place object can be traced back further, ultimately leading to a data structure loaded from a data file named `uk.json`.

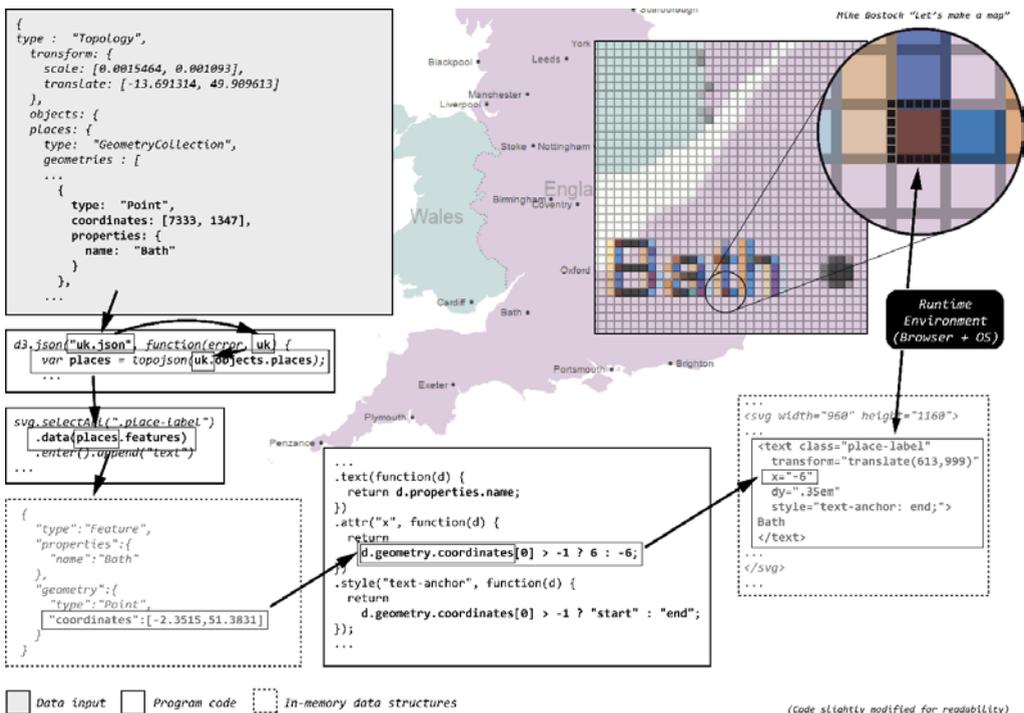


Figure 1: Tracing a single pixel of the map (top right) back through its origins in code. By following the arrows representing transformations backwards, each element of the cartographic output can be traced back to its origin in data and parameters.

Once the transformation of data through code into pixels has been uncovered like this (and the ‘ball of mud’ untangled for inspection), the investigation can continue using other research methods. In modern software projects, code is often stored in a version control system (VCS) that retains, for each line of the program, information on the authorship history plus comments potentially explaining the rationale behind it. This information can now be queried for specific lines of interest to retrieve entry points for an examination of the social processes behind the code, and therefore the map. In the example, the line of code

implementing the label placement algorithm could be investigated and its author contacted for an interview about this decision, the potential alternatives considered etc. Similar investigations could be performed for the origins of the data fed into the program. (See Bates, Lin, & Goodale (2016) for their idea of reconstructing a ‘data journey’, in this case the various steps and transformations involved in delivering a weather datum from the sensor which originally picks it up to the eventual consumers of aggregated weather data online.)

By performing such an ‘ethnography’ of a single pixel of a map, using the source code as an artefact connecting the various components of the map’s ‘socio-technical assemblage’ (Kitchin, 2017), we have uncovered explicit rules and algorithms embedded in the source code that determine the output of the map. In the trivial example of the label placement, an informed observer could have inferred such an algorithm from careful examination of the map image. But other questions of interest that remain opaque to an observer of the map as an image only can be imagined (for example, where do the colours on the map come from? Are they chosen by the programmer or do they represent some statistical attribute of the data?). But even in simple scenarios like the one presented, uncertainty would remain about how exactly the underlying cartographic rules are defined, and there would be no way to assign fine-grained accountability or to assess the historical evolution of such rules if only the map image were taken into account.

A ‘selective close reading’ approach can uncover truth about maps that would remain opaque to an observer of the map as only an image. Using such a method, investigators are supported in (i) finding explicit representations of decisions and algorithms that determine the visual appearance of the map in the code; (ii) tracing back visual output primitives to their origin in data and visualization parameters, both embedded in the code and loaded from external sources, and (iii) being provided with points of access to further artefacts (data files, version control systems, documentation) that can be used to conduct further investigation using social sciences methods.

However, there are serious limitations to the approach presented in this section that make it impractical to apply in many scenarios (especially in the light of the challenges mentioned in the introduction), and which limit the depth of insight gained by manual close reading.

- The method requires a large amount of manual work by an investigator who is proficient both in programming and in cartography, a human resource which often cannot be made available on a large scale for purely analytical tasks.
- While selective close reading may highlight quite clearly the relevant portions of code to an educated observer, for non-programmers the meaning of that code can still remain opaque. For example, it may not be obvious to a non-programmer that the line `d.geometry.coordinates[0] > -1 ? 6 : -6;` has the semantics ‘If the longitude of the given point is east of the 1°W meridian, position the label at six pixels to the right of the point, else at 6 pixels to the left’.
- Some of the transformation processes at work in the program are not visible in the ‘outermost’ layer of code and can be easily overlooked by investigators. Depending on the design of the APIs used to create the map, some transformations may be represented explicitly in the code, or be performed implicitly behind the scenes. (See Ledermann & Gartner (2015) for a discussion of various aspects of cartographic

API design, including such questions of *transparency*.) In our example, note how the **coordinates** entry of the data object changes from **[7333, 1347]** (in the data file, top left) to **[-2.3515, 51.3831]** (in the internal memory representation, bottom left) without this transformation being explicitly represented in the depicted code!

- While in the simple example used for illustration, a reading of the code and a visualization of the results are possible and provide some insight, it remains doubtful whether a close reading approach can be applied to much more complex real-world examples. Furthermore, for commercial and technical reasons, source code is often *obfuscated*, with all human-readable semantics removed from it. Such code, while technically still executable code of equivalent functionality for the computer (and therefore containing cartographic transformations identical to those of its non-obfuscated version), is unreadable by human standards or would make an analysis orders of magnitude more time-consuming.

For these reasons, while selective close reading served to illustrate the idea that cartographic transformations are indeed represented in source code, it can be applied only to singular examples of specific interest. It is unrealistic to expect such an investigation to be applied on any larger scale, let alone to a significant proportion of the maps posted online. Manual investigation of media artefacts is not suited for coping with the challenges of volume and velocity of our age: to engage with even a meaningful proportion of online maps, automated approaches must be considered.

3 Automated reverse engineering of online maps

Whenever we reach the limits of manual labour (as is the case, as argued in the previous section, with the analysis of cartographic programs), it is desirable to look for ways of automation (Alier & Casany, 2017). Is it possible to provide tools to support researchers carrying out such analyses, or even to create fully automatic systems that perform an analysis of cartographic programs autonomously?

Kitchin (2017) largely dismisses the potential of reverse engineering techniques for the analysis of algorithms. Reverse engineering, in his summary, is considered only for ‘black box’ scenarios, because if one had access to the code, direct analysis would be possible and preferable. However, as has been shown in the conclusions of Section 2, such a direct analysis of the source code has severe limitations. A reverse engineering approach based on a ‘black box’, observing only inputs and outputs of an otherwise hidden algorithm, is also limited, as it would allow investigators only to guess at the internal behaviour of the system. In the example used in Section 2, this would mean having access only to the map image and the input data file (both in the upper part of Figure 1) and trying to infer the rules that created the map from that information. Such a ‘black box’ approach must of course be considered for scenarios in which access to the source code is not available. For example, the actual cartographic images of the Google Maps system are rendered by software on Google’s servers, out of reach for a critical analysis of its code. In such cases, all that can be done is to infer cartographic processes from the delivered map images. (A recent cartographic feature that has been added to Google Maps are coloured ‘areas of interest’ in urban areas. Critical

observers have pointed out that a detailed analysis of the algorithms that determine whether an area is ‘interesting’ or not would be highly desirable (Bliss, 2016)).

But for many maps and geographic visualizations on the web, the code is actually delivered to the user in the form of JavaScript to be run in the browser. This code can be analysed not only by a human actor: it can also be analysed by another computer program. The advantage of computer code over most other cultural artefacts is that it has unambiguous, precisely defined syntax *and* semantics in terms of how it is understood and performed by the computer (the ‘level 2’ of code in Galloway’s taxonomy). It is therefore possible to construct a software that, instead of simply executing the code in a given program, records the operations performed at each step for every value in the program. For every output value (for example, a map pixel), such software could provide an interested observer with a complete ‘genealogy’ of how the value has been calculated – similar to the manual analysis proposed above, but this time without any effort of a human investigator, automatically derived from the program code.

The complete technical details of such a system have been elaborated elsewhere (Ledermann, 2018). Our prototype program for cartographic analysis uses the Jalangi framework (Sen, Kalasapur, Brutch, & Gibbs, 2013) and the puppeteer software (Google Inc., 2017) to autonomously run JavaScript-based programs inside a browser and register every calculation. With this system, it is possible to reproduce, for each attribute of each output primitive of a map, a *transformation graph*, representing all inputs and calculations that were performed by the code for the purpose of producing the value.

Figure 2 shows the reconstructed transformation graph of the example from Section 2. In contrast to the snippets of program code used to represent the transformations in the selective close reading approach, the transformation graph has a much more constrained syntax (every ‘node’ in the graph represents a calculation or transformation with a maximum of 2 ‘inputs’). Such syntax should be more intuitive to understand even by non-programmers, and it is also suitable for further processing by software. The automatic extraction of the transformation graph is able to retain the connection between operations in the graph and the location of their implementation in the source code. It is therefore possible to link directly to the secondary information sources that the code provides (variable names, comments in the code, authorship information, comments and history stored in a VCS etc.) through the transformation graph, making the graph an interface that connects the map image, the input data, the parameters of the transformation and the social assemblage that produced the code. A prototypical viewer for such information reconstructed from the code of online maps has been implemented and is currently being used to analyse various case studies of online cartographic practices.

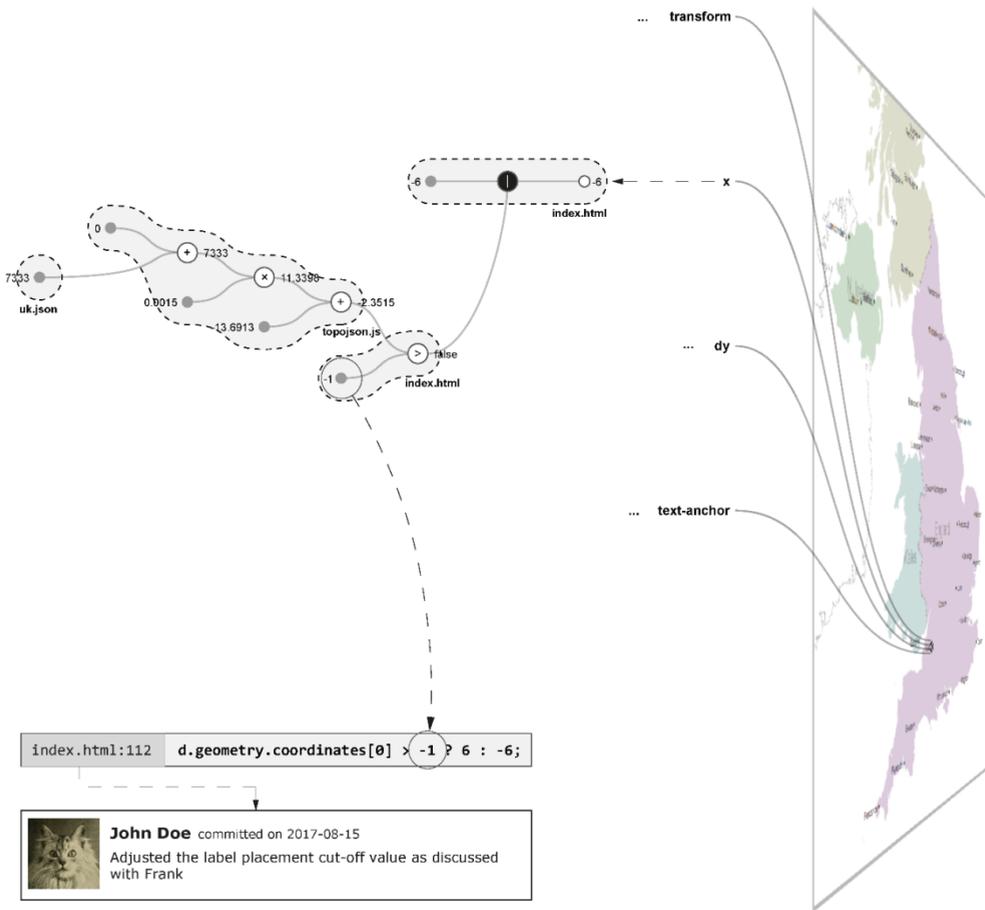


Figure 2: The map from Figure 1, analysed and visualized using automated reverse engineering. The appearance of the label from the map is defined by various attributes, each of which is associated with a transformation graph extracted from the code (here shown for the attribute 'x'). In this case, the value -6 is assigned by a conditional expression (black dot) that checks the condition represented by the remainder of the graph. Each node in the graph can be linked back to its definition in the source code, which in turn can be related to messages in a version control system (lower left). (VCS message fictitious)

4 Outlook: Computer-assisted critical cartography

It is still a long way from the fine-grained representation of the internal calculations of a program provided by the transformation graph to an *understanding* of the processes represented by it or a critique of their appropriateness. It is further unclear whether such an interpretation of the transformation processes involved in the production of a digital map can ever be fully automated. Yet we hope that by extracting a representation of these transformations independent from their underlying syntactic representation in an automated way, a basis can be provided for further research into 'computer-assisted critical cartography'. Since Harley's call for 'deconstructing the map' (Harley, 1989), researchers have

looked for ways to ‘expose in all its unglamorously dishevelled tangle the threads constituting the well heeled image it presents to the world’ (Eagleton, 1986, quoted after Harley). It is hoped that the approach presented here could be a first step towards informing, with evidence derived from engineering methods, not only practical investigation, but also critical and theoretical positions.

On a practical level, the simple syntax of transformation graphs allows for automatic inspection of the patterns of the graph in a subsequent analysis step. Our current ongoing research investigates how to identify patterns for well-known cartographic transformations (projections, classification, generalization etc.), so that the automatic identification of these patterns in online maps becomes possible. The vision is to automatically identify maps that don’t adhere to cartographic best practices, such as using a non-area-preserving projection for statistical maps or using dubious classification or colour schemes that would skew the perception of the depicted data. But also new and creative approaches, not following any well-known patterns, could be identified and highlighted for further inspection by human investigators. As a vision, the construction of a ‘cartographic search engine’, indexing the web or collections of digital maps according to cartographic patterns, seems within reach, and would be a major step towards making such materials more accessible to researchers from various backgrounds and with diverse research questions.

The proposed approach also has the potential to advance our conceptualization of digital maps. So far, a map on the screen can be judged, in the absence of further information, only by its appearance as an image – the colours and other visual variables of the lines, areas and symbols on the map. Using automated analysis of the transformation processes at work to produce the map, we can assess the map as a ‘deep image’, taking into account for each pixel and graphical primitive the technical process that led to its production. Figure 2 can also be seen as an illustration of the concept of the ‘deep image’, rooting each pixel in computational processes and connected digital artefacts. On a general level, maps can be assessed as ‘shallow’, for example if all screen coordinates and colours are simply read from a data file and drawn directly to the screen, or ‘deep’, if sophisticated transformations take place to render the information. While the screen image of a ‘shallow’ map and its ‘deep’ version may be identical, the latter opens itself up to investigation, modification and adaption, while the former leaves its inner workings and their rationale obscured.

It has been shown how the visualization of the transformation graph contributes to answering the call to ‘unpack the full socio-technical assemblage of algorithms’ (Kitchin, 2017). It therefore needs to be investigated whether the proposed approach can also be applied to other types of artefacts in the digital humanities, or even to the very tools that researchers use for analysis and visualization, tools which have, like maps, been criticized as opaque and misleading in some cases (Dobson, 2015). The proposed approach is therefore very much related to the recent call for openness and interdisciplinarity in the digital humanities (Therón & Wandl-Vogt, 2017), with engineers working together with scholars from various disciplines to advance the methods available to improve our understanding of maps and other artefacts produced by software in the digital age.

Acknowledgements

I would like to thank Georg Gartner, Julia Mia Stirnemann and three anonymous reviewers for providing extensive feedback on earlier drafts of this paper.

References

- Alier, M., & Casany, M. J. (2017). The Need for Interdisciplinary Research on Exponential Technologies and Sustainability. In *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 85:1–85:7). New York, NY: ACM. <https://doi.org/10.1145/3144826.3145377>
- Bates, J., Lin, Y.-W., & Goodale, P. (2016). Data journeys: Capturing the socio-material constitution of data objects and flows. *Big Data & Society*, 3(2). <https://doi.org/10.1177/2053951716654502>
- Bliss, L. (2016). The Real Problem With ‘Areas of Interest’ on Google Maps. Retrieved January 26, 2018, from <http://www.citylab.com/design/2016/08/google-maps-areas-of-interest/493670/>
- Bostock, M. (2012). Let’s Make a Map. Retrieved April 7, 2017, from <https://bost.ocks.org/mike/map/>
- Campbell, T. (1989). Understanding Engraved Maps. *The Map Collector*, 46, 2–10.
- Cartwright, W. (2012). Neocartography: Opportunities, issues and prospects. *South African Journal of Geomatics*, 1(1), 14–31.
- Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work* (pp. 1277–1286). New York, NY: ACM. <https://doi.org/10.1145/2145204.2145396>
- Dobson, J. E. (2015). Can An Algorithm Be Disturbed? Machine Learning, Intrinsic Criticism, and the Digital Humanities. *College Literature*, 42(4), 543–564.
- Eagleton, T. (1986). *Against the Grain: Essays 1975 - 1985*. London: Verso.
- Faby, H., & Koch, A. (2010). From maps to neo-cartography. In *Proceedings of the 3rd International Conference on Cartography & GIS*. Nessebar, Bulgaria.
- Foote, B., & Yoder, J. (2000). Big ball of mud. In N. Harrison, B. Foote, & H. Rohnert (Eds.), *Pattern languages of program design* (Vol. 4, pp. 654–692). Boston, Mass.: Addison-Wesley.
- Fuller, M. (2008). *Software Studies: A Lexicon*. Cambridge, Mass.: MIT Press.
- Galloway, A. (2010). Networks. In W. J. T. Mitchell & M. B. N. Hansen (Eds.), *Critical Terms for Media Studies*. Chicago, IL: University Of Chicago Press.
- Google Inc. (2017). puppeteer: Headless Chrome Node API [GitHub Repository]. Retrieved February 1, 2018, from <https://github.com/GoogleChrome/puppeteer>
- Griffin, A. L., Robinson, A. C., & Roth, R. E. (2017). Envisioning the future of cartographic research. *International Journal of Cartography*, 3(sup1), 1–8. <https://doi.org/10.1080/23729333.2017.1316466>
- Harley, J. B. (1989). Deconstructing the map. *Cartographica*, 26(2), 1–20. <https://doi.org/10.3138/E635-7827-1757-9T53>
- Hitzler, P., & Janowicz, K. (2013). Linked Data, Big Data, and the 4th Paradigm. *Semantic Web*, 4(3), 233–235.
- Jenny, B., & Hurni, L. (2011). Studying cartographic heritage: Analysis and visualization of geometric distortions. *Computers & Graphics*, 35(2), 402–411.
- Kitchin, R. (2017). Thinking critically about and researching algorithms. *Information, Communication & Society*, 20(1), 14–29. <https://doi.org/10.1080/1369118X.2016.1154087>
- Ledermann, F. (2016). Initial Findings from Close Reading of Cartographic Programs. In *Workshop ‘Code Loves Maps’, AGILE 2016*. Helsinki, Finland.

- Ledermann, F. (2018). Towards Automatic Extraction of Cartographic Metadata from the Code of Online Maps. Presented at the AGILE 2018, 21st Conference on Geo-information science, Lund, Sweden.
- Ledermann, F., & Gartner, G. (2015). mapmap.js: A Data-Driven Web Mapping API for Thematic Cartography. *Brazilian Journal of Cartography*, 67(5) (Special Issue ICC2015), 1043–1053.
- Mackenzie, A. (2006). *Cutting Code: Software and Sociality*. New York: Peter Lang Publishing.
- Marcotte, E. (2010, May 25). Responsive web design. Retrieved January 23, 2018, from <http://alistapart.com/article/responsive-web-design>
- Monmonier, M. S. (1996). *How to Lie with Maps*. Chicago, IL: University of Chicago Press.
- Roberge, J., & Seyfert, R. (Eds.). (2017). *Algorithmenkulturen: Über die rechnerische Konstruktion der Wirklichkeit* (1st ed.). Bielefeld, DE: transcript-Verlag. <https://doi.org/10.14361/9783839438008>
- Robinson, A. C. (2018). Elements of viral cartography. *Cartography and Geographic Information Science*. <https://doi.org/10.1080/15230406.2018.1484304>
- Roth, R. E. (2016). Rethinking cartography curriculum to train the contemporary cartographer. In *Proceedings of the 6th International Conference on Cartography And GIS* (p. 155). Albena, Bulgaria: Bulgarian Cartographic Association.
- Roth, R. E., Donohue, R. G., Sack, C. M., Wallace, T. R., & Buckingham, T. M. A. (2014). A Process for Keeping Pace with Evolving Web Mapping Technologies. *Cartographic Perspectives*, (78), 25–52. <https://doi.org/10.14714/CP78.1273>
- Sen, K., Kalasapur, S., Brutch, T., & Gibbs, S. (2013). Jalangi: A selective record-replay and dynamic analysis framework for JavaScript. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (pp. 488–498). Saint Petersburg, Russia: ACM.
- Stalder, F. (2016). *Kultur der Digitalität* (1st ed.). Berlin: Suhrkamp.
- Therón, R., & Wandl-Vogt, E. (2017). Overview of the ‘New Trends in Digital Humanities’ Track. In *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 84:1–84:2). New York, NY: ACM. <https://doi.org/10.1145/3144826.3145376>
- Tobler, W. R. (1979). A Transformational View of Cartography. *The American Cartographer*, 6(2), 101–106.