Programmable graphics editors for data visualisation

Mariana Mărășoiu
Computer Laboratory
University of Cambridge
mariana.marasoiu@cl.cam.ac.uk

1. Introduction

There is an increasing interest in using data visualisations to support data analytics. There are principally two types of tools that are being developed for visualisation creation. Firstly, there are programming languages and libraries that support making visualisation through code (e.g. D3.js (Bostock et al., 2011), Processing (Fry & Reas, 2010), ggplot2 (Wickham, 2009)). These tools are very computationally flexible, the user being able to create any type of chart they need. However, they require at least some programming knowledge in order to be successfully used. This makes them unsuitable for the large population of non-programmers who are still interested in data analytics.

The other type of tools is data analytics applications, which usually also have visualisation features available. For example, in Tableau[1] and other similar tools, visualisation creation is the primary activity of the user. This is supported by a drag and drop style interaction, which is often regarded as user-friendly and suitable for users with no traditional programming skills. However, they are limited in the visualisations that can be created: the user can only choose from a fixed set of chart types. Spreadsheet tools like Microsoft Excel[2] have a similar issue: they are computationally flexible, but even more limited in their visualisation capabilities.

My research aims to explore the gap between these two types of tools by looking at ways in which end user programming tools can provide the flexibility of textual programming environments for data analysis through visualisations.

In a research abstract accepted at the VL/HCC Graduate Consortium (Marasoiu, 2016), I am discussing my research plans and this project in the context of existing information visualisation tools. At the PPIG Doctoral Consortium, I am looking to discuss several theoretical issues which underlie my research project. Towards this, I give in the next section a sketch description of how a data visualisation tool that bridges this gap might look and in the last section a set of questions as a starting point for the discussion at the consortium.

2. End-User Programming for Data Visualisations

My proposed solution for creating data visualisation is a system similar to graphics editors, where the user is able to create, manipulate and combine shapes in order to build a complex graphical image. The user can parametrise the dimensions and other properties of those shapes with actual data values, creating an image that is backed by data.

There have been several previous attempts at achieving this. One of the first was Myers' Gold prototype (1994), where data was displayed in a spreadsheet and the user dragged and dropped ranges of values from the spreadsheet onto shapes. The system then generated new shapes and aligned them on the canvas using rule-based inference. Victor (2013) demonstrates a system for drawing dynamic images that lets the user manipulate shapes on a canvas and parametrise their properties with existing data. As the user creates a shape, she specifies how the shape's dimensions and position correspond to the first data point. The system records the steps used to create the shape, which the user can then select and create a loop out of them. The system makes explicit all computations: the user can observe how the visualisation is created by stepping through the loop for each data point.

---

[1] Tableau, Tableau Software, http://www.tableau.com/
[2] Microsoft Excel 2016, Microsoft Corporation, https://products.office.com/excel

Another set of tools that can be drawn upon is that of constraint-based drawing tools. Probably the most well known example is Sutherland's Sketchpad (1963), and more recently Wybrow's Dunnart (Dwyer et al., 2009). Further, constraint-based drawing is also available in a more lightweight form in commercial graphics editors through dynamic guides: the user can resize/move shapes and snap them to relevant sizes/positions on the canvas that align with other shapes (e.g. Adobe Illustrator[3], CorelDRAW[4]). This alignment is temporary - moving a shape doesn't affect other shapes aligned to it. A more permanent constraint is found in diagramming tools like Microsoft Visio[5], where shapes can be linked to other shapes through connectors. Including some of these techniques into a data visualisation system could be a useful activity, as each shape of the chart is in a visual relation to the others (e.g. all bars in a typical bar chart are aligned horizontally at the bottom).

Building on this previous work, I'm looking at ways in which spreadsheets and graphics editors can be merged in order to create data visualisations. The following series of sketches describes how a user might interact with such a system.

The user selects the geospatial coordinate system and then draws a circle on the map (the orange circle in Fig. 1). She then drags and drops values from the spreadsheet (lat, lon, GDP) onto the various properties of the circle (radius, x and y position).

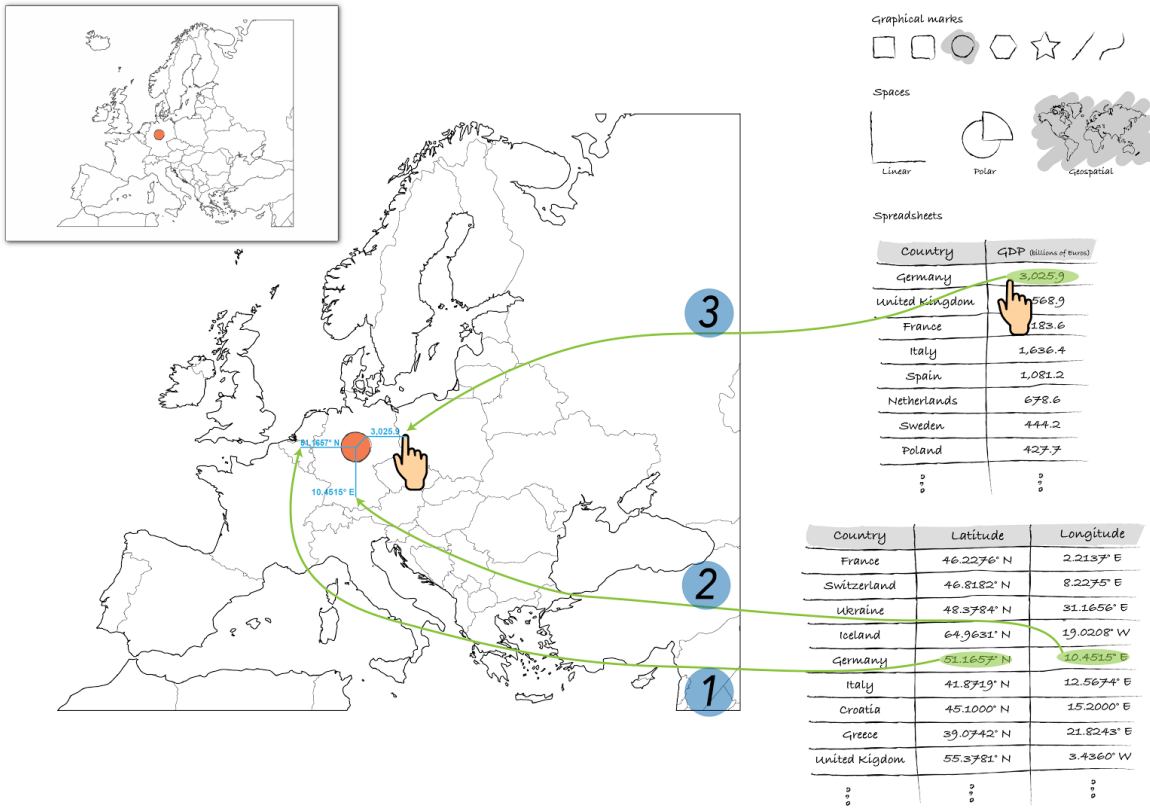

Figure 1. A sketch of the proposed interface for building data visualisations in a graphic-editor style. On the right side, there are a set of tools available to the user: graphical marks, coordinate systems and data in spreadsheet form. In the top-left there is a preview panel which shows the user how her actions to a single data point would look if applied to all data points. The canvas is where the user creates and interacts with the visualisation. With numbered green lines are shown the possible actions of a user: dragging data from the spreadsheet onto the attributes of the orange circle on the canvas.

[3] Adobe Illustrator CC, Adobe System Incorporated, http://www.adobe.com/Illustrator
[4] CorelDRAW, Corel Corporation, http://www.coreldraw.com/
[5] Microsoft Visio, Microsoft Corporation, https://products.office.com/en-gb/visio

The preview window in the top-left corner is showing at each step how the visualisation might look if the system generates circles for each data point (Fig. 2). Notice that after binding the latitude value (1), the circles in the preview have the x position set, but not any other attributes. Similar when connecting the latitude (2) and then the radius (3).

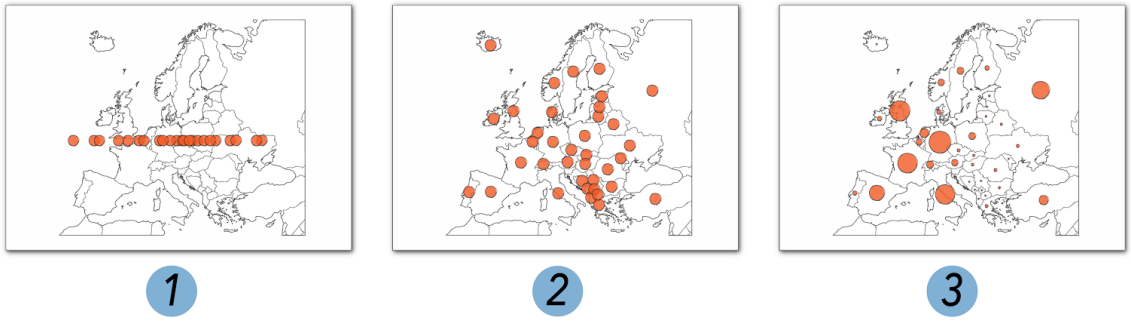

Figure 2. The evolution of the preview panel according to the actions of the user. (1) The user has dragged the longitude value from the spreadsheet onto the "x" attribute of the orange circle. The system generalizes this to all data points and disperses them horizontally. (2) The circles are dispersed on the y axis when the user has dragged the latitude value from the spreadsheet onto the "y" attribute of the orange circle. (3) The orange circles are have different radiuses when the user drags the GDP value onto the radius of the orange circle they're working with.

During this time, the main canvas only has one circle corresponding to one data point which the user edits. When the user is happy with how the preview looks, she can click on it and the main canvas now displays all the generated points (Fig. 3).

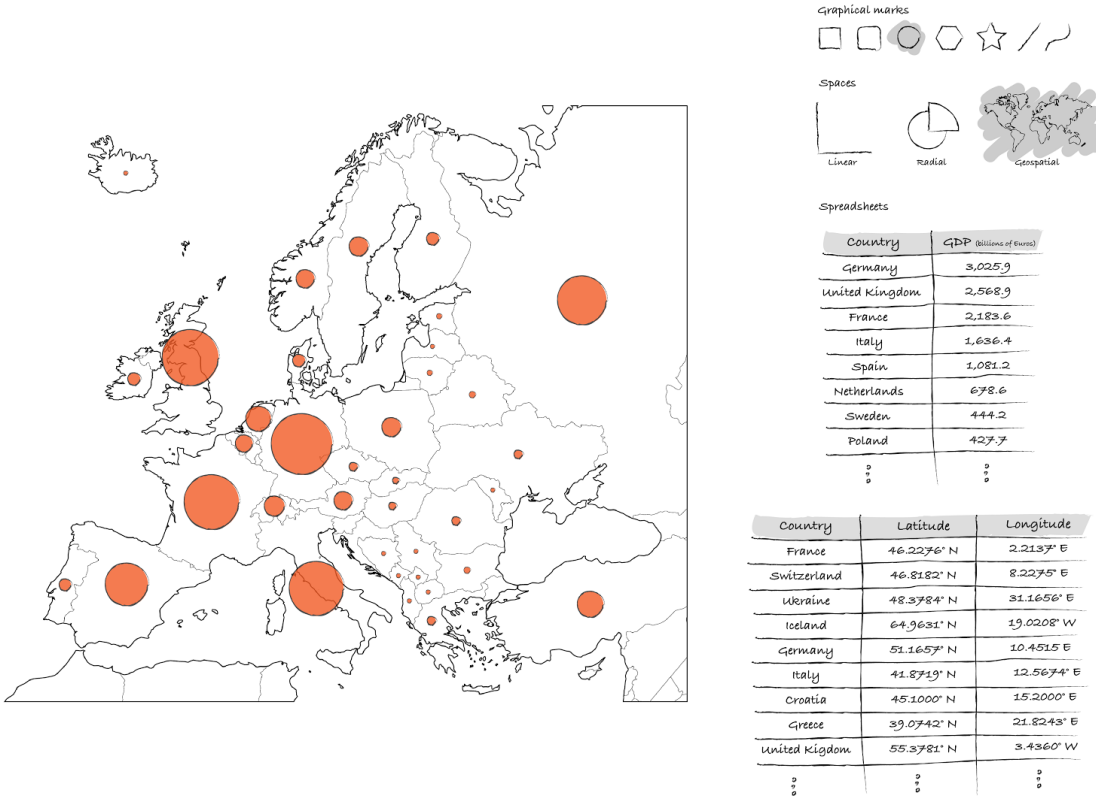

Figure 3. The user is happy with how the preview looks and accepts the system's proposed generalization. All the other shapes are generated on the main visualisation canvas.

The user can continue adjusting the visualisation by, for example, adding labels, changing colours, emphasizing some points, adding more data. However, it is also important to allow modification of a single point and not all (e.g. to emphasize interesting findings). As I will also discuss in Section 3.2 and in 3.3, combining group actions with exceptions to these actions is an open research question.

Further, I am particularly interested in the connection between the shapes and the data points. In existing tools, the user cannot create a visualisation without a dataset. But users sometimes find themselves needing to think about the data analysis and to create visualisations in the absence of data (Mărășoiu et al., 2016). A tool in which users can create visualisations with some or even no data may prove useful. The details of this interaction still need to be worked out, but we can imagine for example the user creating several circles to start with and arranging them on the map as an indication of how she expects the visualisation to look like. When data becomes available, she can then bind the real data to the visualisation. The visualisation then updates to reflect the real data.

3. Theoretical concerns

3.1 Programming by demonstration, by example or with examples?

One of the main theoretical issues that needs clarifying is what style of programming such a system is supporting.

The closest description to such a system would be that of a programming by example (PbE) or by demonstration (PbD). Either term is used to refer to programming by direct manipulation where the user interacts with examples and concrete artefacts of the program. However, Blackwell (2006) argued that there is a difference between the two: "In programming by demonstration, some abstract notational conventions are displayed alongside the data of interest, and the user directly manipulates that notation in addition to manipulating the data." An example would be that of a system which creates generalized programs from a set of recorded user actions and which allows the user to edit the generated program.

Further, "in programming by example, the user provides several examples of the required program behavior, and the system applies an inference algorithm in order to infer a generalized program that will operate in accordance with the user's intentions in other situations not yet seen." (Blackwell, 2006). Here, an example would be that of a system which is provided with a set of text examples and infers a regular expression that matches the text (Blackwell, 2001; Church & Blackwell, 2008).

The system for data visualisation described in the previous section doesn't precisely match either description. Compared to PbD, the system does not need to directly display the notational conventions or the program that is generated from user's actions, the visualisation can be the only output. Further, compared to PbE, there may not be the need for using inferencing techniques - the system simply generates more shapes for each data points following the example of the first data point which was parametrized by the user.

Is there an alternative theoretical framing that is a more generative description of this kind of system? Perhaps *programming with concrete examples* is a more appropriate term, as was done in e.g. Smalltalk (Goldberg & Robson, 1983). What is the relationship between programming with concrete examples and programming by example?

3.2 Data binding rules, with exceptions

When creating data visualisations, especially for presentation purposes, appearance is important. It is thus useful for the user to be able to adjust some data points in order for the graphic to look good and better convey the message (e.g. moving the label of a circle so that it doesn't overlap with others). This means that the system needs to retain the same flexibility of normal graphics editors, whilst maintaining connections to the data.

Allowing such exceptions to the layout creates hidden dependencies (Green & Petre, 1996), as the data that parametrizes the shape will have both general and particular components. What are the ways in which interfaces could express such exceptions in a way that is easy to understand for end users?

3.3 Progressive abstractions

When building visualisations directly from shapes, the user interacts with concrete objects: each shape has a set of attributes like width, height, position, color etc. Once all the shapes in the chart have been generated, the user will likely think about the chart as a whole in the interactions between the chart and the rest of the canvas. However, the interface should still allow the user to think about the individual shapes, for example, when making exceptions to the rules that created the chart.

In Cognitive Dimensions of Notations terminology, a large number of similar shapes would have the problem of 'repetition viscosity' (Green & Blackwell, 1998), where a single conceptual action (e.g. arranging the labels of the visualisation) requires many small actions (moving the label of each circle). The usual solution for reducing viscosity is to introduce an abstraction - for example, once the pie is created, the system would work with it as a whole. Unfortunately, this may make editing some parts of the chart difficult or even impossible.

Further, as mentioned in Section 2, a graphics-oriented data visualisation tool would benefit from research in the constraint-based layout systems for creating diagrams. These constraints are in fact abstractions over the shapes, implemented at different levels of temporality. For example, a transitory abstraction would be the dynamic guides in graphics editors like Adobe Illustrator which appear only when a shape is resized. A more permanent abstraction would be the ability to link shapes with connectors (which remain connected when shapes are moved around) in diagram tools like Microsoft Visio.

Which of these levels of abstraction and temporality is appropriate for a data visualisation system? How would intermediate levels of abstraction for such a tool look? Are there any techniques for making progressive abstraction easy for end users?

Answers to the questions above would help the next steps of my research. Answering the first question would provide a framework for my future research and would help me better situate my work within existing research, whilst answers to the other two questions would offer guidance in the design and implementation of the proposed system.

4. References

Blackwell, A. F. (2001). SWYN: A Visual Representation for Regular Expressions. In H. Lieberman (Ed.), *Your Wish Is My Command: Programming by Example* (pp. 245–270). Morgan Kaufmann.

Blackwell, A. F. (2006). Psychological Issues in End-User Programming. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), *End User Development* (Vol. 9, pp. 9–30). Dordrecht: Springer Netherlands.

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2301–2309.

Church, L., & Blackwell, A. F. (2008). Structured Text Modification Using Guided Inference. In *Proceedings PPIG 2008, 20th annual workshop of the Psychology of Programming Interest Group*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.585.6371&rep=rep1&type=pdf

Dwyer, T., Marriott, K., & Wybrow, M. (2009). Dunnart: A Constraint-Based Network Diagram Authoring Tool. In I. G. Tollis & M. Patrignani (Eds.), *Proceedings of the 16th International Symposium on Graph Drawing (GD'08)* (pp. 420–431). Springer-Verlag.

Fry, B., & Reas, C. (2010). Processing. Retrieved from https://processing.org/

Goldberg, A., & Robson, D. (1983). *Smalltalk-80: The Language and Its Implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Green, T., & Blackwell, A. (1998). *Cognitive Dimensions of Information Artefacts: a tutorial*. Presented at the Tutorial session at British Computer Society conference on Human Computer Interaction HCI'98. Retrieved from https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: a "cognitive dimensions" framework. *Journal of Visual Languages & Computing*, *7*(2), 131–174.

Marasoiu, M. (2016). End User Programming of Visualisations. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Cambridge, UK.

Mărășoiu, M., Blackwell, A., Sarkar, A., & Spott, M. (2016). Clarifying hypotheses by sketching data. In *EuroVis 2016 - Short Papers*.

Myers, B. A., Goldstein, J., & Goldberg, M. A. (1994). Creating charts by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 106–111). New York, NY, USA: ACM.

Sutherland, I. E. (1963, January). *Sketchpad: A man-machine graphical communication system*. Massachusetts Institute of Technology. Retrieved from https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf

Victor, B. (2013). *Drawing Dynamic Visualizations*. Retrieved from https://vimeo.com/66085662

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer-Verlag.