EG UK Computer Graphics & Visual Computing (2019)
G. K. L. Tam and J. C. Roberts (Editors)

Towards a tool for the creation of micro-visualisations

J. R. Jackson[†1] and P. D. Ritsos[‡1] and J. C. Roberts[§1]

[1]Bangor University, UK

Abstract

As the every day use of mobile and small screen devices becomes more common, it is necessary to explore how we can visualise data effectively in small design spaces. These screens are often used in situations where it is necessary to convey information in a concise, readable, reliable and visually appealing way. Our work focuses on the design and development of a tool to facilitate the creation and manipulation of new micro-visualisations. The results show that the tool is suitable for creating large number of outputs quickly and efficiently.

CCS Concepts

• ***Human-centered computing*** → ***Visualization;***

1. Introduction

In many settings users want to see a summary view of their information. For example, sports users with smart watches, desktop users who wish to receive announcement summaries of systems they monitor, or notifications on a smart phone, are all situations which involve small screens and the user needs to see summary information. Glyphs provide an ideal structure to summarise multivariate data. Glyphs are small, compact, graphical pictures, which can demonstrate quantitative data, and encode many variables, in a small area. For instance, weather map glyphs encode a range of parameters including wind direction, speed, temperature etc in one multifaceted arrow.

Most maps used for navigation employ glyphs. In addition, some of the glyphs share similar design features to another other. Where, for instance, several glyphs can share similar design traits. For instance, churches have similar designs; those with spires have a slightly adapted design to those without. Consequently, it is useful to create several glyph designs for different data parameters. Much like human children look similar to their parents, or siblings, so we could create glyphs that appear similar to each other. Glyphs that have similar design traits (yet are different by some means) may then display data that has some overlap or share similar properties.

We have been working with a health care provider who has requested that the summary information is displayed differently for several purposes. For example, a nurse may want to have a glyph design for self care, a different one for minimal care patients and another for severe patient care. Potentially every patient has a different glyph pattern, but all the self caring patients would be easily recognisable because their design traits are similar. In other situations, users may wish to create their own personalised glyph. Much like a logo is created specifically for a company, so a user would want to create their own glyph design.

But designing a glyph is not an easy task for a user to achieve. If requires a user to construct the geometry, ascertain how to map the data into individual parts, and then decide how to render it. Consequently, we have been investigating how we can create tools and methods, to help users more easily create glyphs and unit-based visualisations.

In this short paper we introduce a tool which implements a path-based algorithm for generating deterministic glyph visualisations and present several results of the algorithm. We focus on the tool, explaining our requirements (Section 3), different prototype implementations (Section 4), how to use the tool (Section 5) and evaluation and results of using our tool (Section 6).

2. Background Related Work

Micro-visualisations are a type of glyph, according to Borgo et al. [BKC*13] glyphs can depict data attributes using a collection of visual elements in a standalone context, this is included in a variety of definitions. Our use of the term *micro-visualisation* allows for less ambiguity in our output type as well as being more descriptive in terms of their small screen or design space usage.

Currently there are many methods for the production of visualisations. These methods range from declarative tools (such as Vega Lite [SMWH17], D3 [BOH11] and ProtoVis [BH09]) to interface based applications such as (Polaris [STH02], Tableau). Users are

† email: j.r.jackson@bangor.ac.uk
‡ email: p.ritsos@bangor.ac.uk
§ email: j.c.roberts@bangor.ac.uk

also able to create visualisations using vector based software such as Adobe Illustrator and Data Illustator [LTW*18]. Including visualisations designed by sketching [RHR16], there are a wide variety of options available. These options tend to put the user at the centre of the process and lead to one off visualisations which are often predesigned and prepackaged. We introduce a design centric tool which is closer to glyph generation strategies [BKC*13] than user-centric visualisation design.

In order to facilitate the easy generation of visualisations we look to the world of simple programming with applications such as Scratch [RMMH*09] and Geenfoot [Köl10]. These systems adopt drag and drop interaction which obfuscates any code and gives novice users a better understanding of building an application. We use a similar design principle in our interface design.

3. Usage Requirements

With the discussed ubiquity of data both personal and other it is necessary to create a wide range of usable and desirable visualisations. Creating these visualisations can be a slow, drawn-out and difficult process. As such we define a series of requirements which will allow for the rapid prototyping and creation of such designs:

- **Rapid prototyping.** Design of novel and interesting visualisations often requires the creation of a large number of prototypes. Using traditional methods, such as sketching or using graphics tools can be a limiting process which takes a long time. We propose that any tool which aims to aid visualisation design and which must have an emphasis on rapid prototyping.
- **Ease of use.** Many tools exist for the creation of visualisations, using tools such as D3.js, Photoshop and Vega often requires a steep learning curve. The tool which we create must be simple and easy to use without prior knowledge of design or formal visualisation methods. We can aid this by looking at the world of visual programming [RMMH*09].
- **Sensible constraints.** We aim to provide users with a blank canvas, such that their designs are not overly limited or directed. We do, however, recognise that a complete sand-box for design would not necessarily allow for a quick and creative process. The application must tread the fine line between constraint and freedom. This boundary will allow users a very large scope for design and iteration while ensuring that the process is finite and that the outputs are sensible.
- **Easy refinement.** Once visualisations are created we aim to allow users to easily refine those that have been created. Often termed tweaking, the act of refining and making small adjustments is essential to the design process.
- **Creation of usable visualisations.** While the main point of the application is to aid the design and development of visualisations, we must ensure that said visualisations are usable when data is applied. Ensuring that the sensible constraints discussed above and allowing users to easily apply data to generated visualisations will ensure that the vast majority of created graphics are fit for purpose.
- **Usable output.** The visualisations are intended for use on mobile phones or smart watches as well as within desktop applications. As such it is important that the output from the tool is usable in these contexts. We propose to output Scalable Vector Graphics (SVG) from the tool. These are easily placed within target applications and their features can be adjusted using simple properties or external styles. Further to this simple output the tool will allow output of properties which can be used directly with an implementation library.

These requirements allowed us to create a usable and reliable tool which aids in the creation of interesting, usable and useful visualisations. The rest of this paper we (1) explain how have designed and created a browser based application using modern standards such that it can be used on a wide range of modern browsers. (2) Describe how we have created an underlying library, which is also be deployable across a wide variety of systems and architectures, to implement the path-model. And (3) describe how users can use the tool, and present an evaluation of the micro visualisation creator.

4. Implementing the tool

The tool has been developed as a browser based application using ReactJS and employing SVG technology. In order to create visualisations we employ a path-model [JRR18]. This model describes visualisations using a path. The path is defined as a finite collection of point pairs representing each data point. Within each pair of points visualisation elements are placed. These elements are adjusted according to the data value within. These adjustments can be using any of Bertin's visual variables [BB73]. This model uses five stages in generating each design:

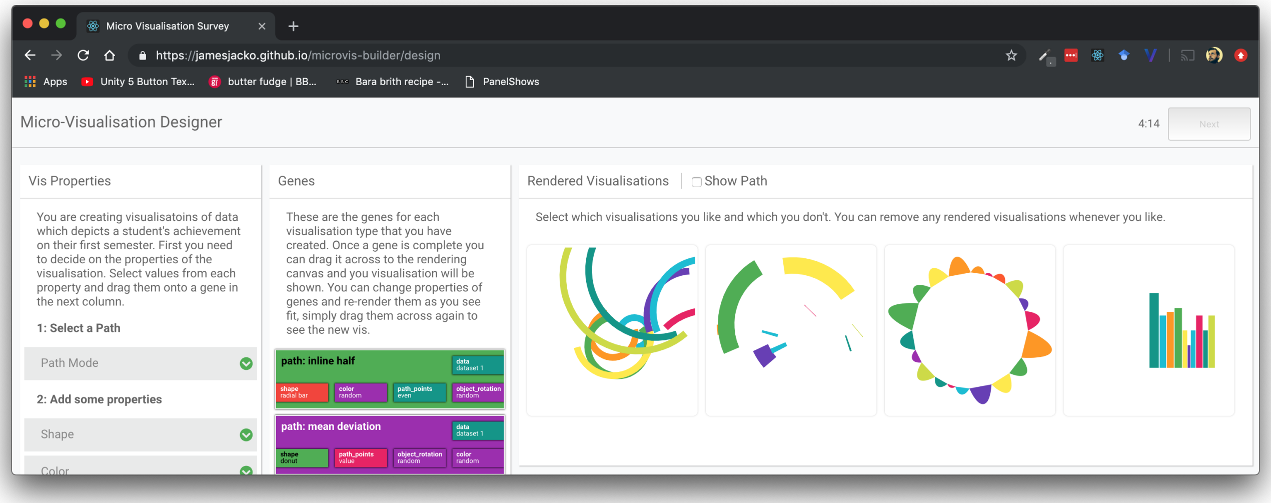
1. We generate a path, where each data point is represented by a pair of points (path section). The path defines the principle shape of the micro-visualisation.
2. We define individual objects, constraining them by an envelope.
3. Visual elements are placed along the path, within the envelope;
4. Data is mapped onto retinal variables [BB73] (e.g., shape, colour, orientation), are adjusted according to input data, placed on the path.
5. The visualisation output (in SVG), rendered on the design space.

In the case of this implementation, scale, colour and rotation are the aspects of the elements which can be adjusted by the tool, these constraints allow users to create a wide variety of visualisations without getting overwhelmed.

4.1. Implementation

We used an Agile method to design the tool; we started by sketching the ideas and then developed several prototypes. We explain our process below.

Sketching In order to design the tool an extensive Five Design Sheet method was used [RHR17]. With this method five sheets of large paper are used to explore and record different design ideas. The first sheet allows many ideas to be investigated; the middle three sheets allow three ideas to be further developed, and the final sheet provides a space for the realisation design to be elaborated. Throughout this process we focused on the user requirements of rapid prototyping, easy of use etc. We wanted a design that required little to no keyboard input, and subsequently we concluded with a design that uses drag-and-drop, based around the path-model, as described above.



You are creating visualisatoins of data which depicts a student's achievement on their first semester. First you need to decide on the properties of the visualisation. Select values from each property and drag them onto a gene in the next column.

These are the genes for each visualisation type that you have created. Once a gene is complete you can drag it across to the rendering canvas and you visualisation will be shown. You can change properties of genes and re-render them as you see fit, simply drag them across again to see the new vis.

Select which visualisations you like and which you don't. You can remove any rendered visualisations whenever you like.

Figure 1: *The MicroVis Builder, this web based application allows users to drag properties from the left column onto genes in the centre, then the visualisation can be rendered on the right.*

Prototype One The first prototype was a point and click interface which allowed users to select visualisation properties using checkboxes. Visualisations were created by importing JSON which contained image space properties and data. Once defined each visualisation was created by drawing to an HTML canvas context. Each visualisation could be exported as an SVG, PNG or JPEG. This first prototype proved problematic as the selection of properties was far from intuitive, the requirement of JSON did not meet the ease of use standard previously discussed. The major issue with the first prototype was the output as it did not output files that could be further manipulated in another context. Work was done to try and make more usable SVG outputs but this proved difficult and highlighted the limitation of the current implementation. This first prototype was constructed with HTML5 Elements, CSS and Vanilla JavaScript.

Prototype Two We performed in-house testing of the first prototype. The evaluation demonstrated that the current interface was too complex, and users did not understand what parameters they could change to alter the shapes of a micro visualisation. Subsequently, we conducted a further design and sketching period, with the goal to simplify the interface, and improve the usability of the final output. We also realised that the current implementation language restricted us from simplifying the interface. Consequently, we implemented the next prototype tool using ReactJS. We chose ReactJS because it allows for the streamlining of web application development as well has having native support of SVGs. The intention was to allow users to quickly and easily create a large number of visualisations, this virtual DOM management will allow the tool to only update the necessary DOM nodes. We separated the prototype into two discrete parts, first the path visualisation library, and second the MicroVis builder, as follows.

The path visualisation library. We have developed a visualisation library that implements the path-model. The library can be used to develop micro visualisations. The developer can use it to take properties and data and render SVG elements into a page or application. The library can be used independently of the MicroVis builder. One of the outputs of the MicroVis builder is to save configuration files (genes) that describe the design of specific visualisations. The library creates visualisations based on a path. There are four parts to the visualisation creation within the library, 1) Path generation – a series of point pairs are generated inline with the number of data points; 2) Object selection and placement – Objects types are selected and placed on the path; 3) Object adjustment – Objects are adjusted to represent the data using any of Bertin's visual variables; 4) Visualisation Realisation – The visualisation is rendered as an SVG to be used within another context such as an app or web page.

The MicroVis Builder is an interface which allows the creation of visualisations using property collections described as Genes. This interfaces uses drag and drop commands to create and render visualisations. Data used in the visualisations can be selected from a collection of pre-installed datasets as well as being able to upload data in JSON format. The application exports SVG, PNG and/or JSON configuration data for the Path Visualisation Library. The tool uses the Local Storage API to save details of users' visualisation. The intention is to allow completely anonymous use of the tool to avoid unnecessary Data privacy and GDPR concerns.

5. Using the Tool

This tool is hosted as an online application and available to all modern browsers. There is no login or registration requirement for users. They simply visit the URL and create visualisations. The tool has a memory such that it saves created visualisation information to the user's browser. An overview of the tool can be seen in Figure 1. The MicroVis Builder uses drag and drop and properties are dragged onto genes and genes are dragged to the rendering stage.

A visualisation is created by selecting properties from the left section (Figure 1 left column). These properties are a small set of the properties available within the Path Visualisation Library. Users first select a Path Mode, this is the underlying path that the visualisation will be created along. To simplify the user-interface in this prototype we predefine six paths, including straight line, ring and random paths. Once a path is chosen users can choose from a series of properties for the visualisation elements. These properties are the shape of the visualisation elements such as circles, triangles and squares; the colour of the elements, based on value, random, monotone, single colour or as prescribed by the dataset; the distance between each path point these can be uniform, random or as a function of the value; rotation of the objects including none, random, value based or mean based. After the visualisation properties have been chosen the user can now select a dataset from the presets or upload their own. Property types are colour coded to give visualisations a colour identity. Each of these properties can be dragged across to the gene area (Figure 1 centre column) where new genes are built or existing ones updated. The gene builder section allows users to create, update and delete genes as well as inspect them. Each gene can be clicked to give the user more readable information. Genes can be copied to allow users to make small adjustments while maintaining the original settings. The Gene builder panel is also able to generate random genes to help to give users a new perspective or starting point.

6. Evaluation and Results

In order to evaluate the tool we asked participants to spend five minutes creating visualisations with the tool. Participants were sourced through social media and each participant was asked to watch an instructional video prior to starting. Users created visualisations using one of three datasets of varying sizes (4, 8, 20 data points) once created they were also asked to like or dislike their creations. Users could only progress to the next part of the study one five minutes had elapsed (they were not forced to move on and were able to continue once the timer had elapsed). The second part of the study involved a System Usability Scale survey as well as a survey of three open ended questions.

A selection of user results can be seen in Figure 2. The 18 participants created a total of 163 visualisations. The minimum number of visualisations created was 4 and the maximum 15. This means that, on average, just over 9 visualisations were created per participant within five minutes. The average time spent designing was 356.5 seconds with a minimum of 302.2 seconds and a maximum of 689.3 seconds. Using this data we can infer that users of the model are able to create a visualisation every 39.37 seconds. Of the 163 visualisations that were created 89 were liked and the remaining 74 disliked. Due to the nature of the design process we expect to see a high number of disliked visualisations as a decision is made when updating and re rendering genes. The binary choice between like and dislike forced participants to make clear decisions over the rendered visualisation and whether amendments of the gene were

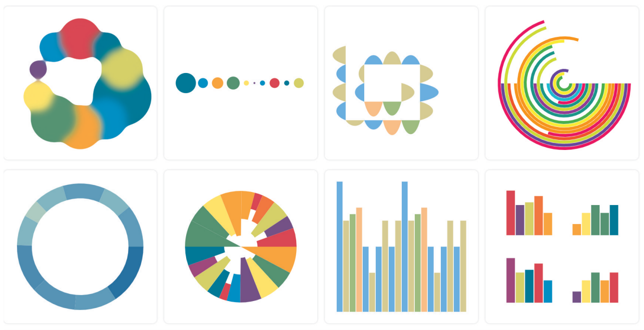


Figure 2: *A selection of user outputs from the user study.*

needed. 157 of the visualisations were created using a unique combination of gene parameters.

The SUS resulted in a mean score of 75.42 indicating a 'good' score according to Bangor et al. [BKM09]. In the open ended questions, users were largely complementary stating that the tool was *"simple to use"* or *"very easy to use"* and that it could *"could create some interesting designs very fast"*. Some critical feedback was also registered one participant stated *"I would like visual description of what each property would do to my visualisation."* and *"when there were many visualisations, knowing which gene corresponded to which visualisation was tricky"*.

7. Conclusion

We have designed and implemented library and a tool (MicroVis builder) to allow users to quickly and easily create micro visualisations. This tool uses drag and drop interaction to build path based genes which are then able to be rendered and tweaked. Once the visualisations are created they can be downloaded in three format, as JSON properties, as SVG or as high DPI PNG files. Overall feedback of the tool was positive as it received a *good* rated SUS score as well as receiving positive comments.

This work is part of ongoing research, and indeed further work is also needed to explore the effectiveness and suitability of the outputs in their intended contexts. In the future we would like to extend the functionality of the tool to allow users to input their own data and use the visualisations in-situ such as on mobile applications or smartwatch faces.

Acknowledgements

Knowledge Economy Skills Scholarships (KESS 2) is a pan-Wales higher level skills initiative led by Bangor University on behalf of the HE sector in Wales. It is part funded by the Welsh Government's European Social Fund (ESF) convergence programme for West Wales and the Valleys.

References

[BB73] BERTIN J., BARBUT M.: *Sémiologie graphique: les diagrammes, les réseaux, les cartes*. Gauthier Villars, 1973. 2

[BH09] BOSTOCK M., HEER J.: Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov 2009), 1121–1128. doi:10.1109/TVCG.2009.174. 1

[BKC*13] BORGO R., KEHRER J., CHUNG D. H., MAGUIRE E., LARAMEE R. S., HAUSER H., WARD M., CHEN M.: Glyph-based visualization: Foundations, design guidelines, techniques and applications. In *Eurographics (STARs)* (2013), pp. 39–63. doi:10.2312/conf/EG2013/stars/039-063. 1, 2

[BKM09] BANGOR A., KORTUM P., MILLER J.: Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies 4*, 3 (May 2009), 114–123. 4

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: DÂş data-driven documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec 2011), 2301–2309. doi:10.1109/TVCG.2011.185. 1

[JRR18] JACKSON J., RITSOS P. D., ROBERTS J. C.: Creating Small Unit Based Glyph Visualisations. In *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2018), Berlin, Germany* (Oct. 2018). 2

[Köl10] KÖLLING M.: The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE) 10*, 4 (2010), 14. doi:10.1145/1868358.1868361. 2

[LTW*18] LIU Z., THOMPSON J., WILSON A., DONTCHEVA M., DELOREY J., GRIGG S., KERR B., STASKO J.: Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2018), CHI '18, ACM, pp. 123:1–123:13. doi:10.1145/3173574.3173697. 2

[RHR16] ROBERTS J. C., HEADLEAND C., RITSOS P. D.: Sketching designs using the five design-sheet methodology. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan 2016), 419–428. doi:10.1109/TVCG.2015.2467271. 2

[RHR17] ROBERTS J. C., HEADLEAND C. J., RITSOS P. D.: *Five Design-Sheets: Creative Design and Sketching for Computing and Visualisation*. Springer, 2017. 2

[RMMH*09] RESNICK M., MALONEY J., MONROY-HERNÁNDEZ A., RUSK N., EASTMOND E., BRENNAN K., MILLNER A., ROSENBAUM E., SILVER J. S., SILVERMAN B., ET AL.: Scratch: Programming for all. *Commun. Acm 52*, 11 (2009), 60–67. 2

[SMWH17] SATYANARAYAN A., MORITZ D., WONGSUPHASAWAT K., HEER J.: Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan 2017), 341–350. doi:10.1109/TVCG.2016.2599030. 1

[STH02] STOLTE C., TANG D., HANRAHAN P.: Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (Jan 2002), 52–65. doi:10.1109/2945.981851. 1