Path-based Design Model
for Constructing and Exploring Alternative Visualisations

James Jackson ⓘ, Panagiotis D. Ritsos ⓘ, Peter W. S. Butcher ⓘ and Jonathan C. Roberts ⓘ

Fig. 1: The Genii visualisation designer that demonstrates our flowpath model. Individuals define their own path or choose predefined flowpaths (left panel), drag and drop the visualisation properties into the gene panel (middle), which are rendered onto the gallery (right). Users can either create a new gene which adds a new image to the gallery, or edit parameters (through drag and drop) to adapt current visualisations. Crafted visualisations can be exported and used in other applications.

Abstract—We present a path-based design model and system for designing and creating visualisations. Our model represents a systematic approach to constructing visual representations of data or concepts following a predefined sequence of steps. The initial step involves outlining the overall appearance of the visualisation by creating a skeleton structure, referred to as a flowpath. Subsequently, we specify objects, visual marks, properties, and appearance, storing them in a gene. Lastly, we map data onto the flowpath, ensuring suitable morphisms. Alternative designs are created by exchanging values in the gene. For example, designs that share similar traits, are created by making small incremental changes to the gene. Our design methodology fosters the generation of diverse creative concepts, space-filling visualisations, and traditional formats like bar charts, circular plots and pie charts. Through our implementation we showcase the model in action. As an example application, we integrate the output visualisations onto a smartwatch and visualisation dashboards. In this article we (1) introduce, define and explain the path model and discuss possibilities for its use, (2) present our implementation, results, and evaluation, and (3) demonstrate and evaluate an application of its use on a mobile watch.

Index Terms—Path-based design, Visualisation Design, Alternative Visualisations

---

◆

---

1 INTRODUCTION

Alternative visualisations are useful but challenging to create. Employed throughout the visualisation domain, they range from sketched planning to explore potential design solutions, to multiple views to explore alternative viewpoints. They offer different perspectives, inspire creativity, validate insights, and help communicate varied narratives. When people set out to create a new visualisation, they often explore numerous design ideas, making many quick sketches on paper. For example, by adhering to methods such as the Five Design-Sheets approach, people produce numerous quick sketches to outline their concepts [73]. They draw a horizontal line to represent an axis, and place three or four rectangles above it to depict a bar chart. Draw two perpendicular lines to represent an X and Y axis, and add a few circles within the space to depict a scatter plot. Or draw a simple rectangle to

• *J. Jackson was with the School of Computer Science and Engineering, Bangor University, UK. Now with ExaDev.*
• *P.D. Ritsos, P.W.S Butcher and J.C. Roberts, are with the School of Computer Science and Engineering, Bangor University, UK. E-mail: {p.ritsos;p.butcher,j.c.roberts}@bangor.ac.uk.*

denote a computer monitor.

Individuals are sketching skeletal illustrations to represent various visualisation concepts. They begin by outlining shapes such as rectangles or curves and straight lines to capture the fundamental structure and form of the object being depicted. Similar skeletal outlines are employed in other areas of design. This approach reflects a fundamental principle in figure drawing: starting with the essence or skeleton of the subject before adding details. Frank Reilly explained this process as starting to draw the human form from its "line-of-action" [23, 50]. It involves drawing lines that represent the primary movement or flow of the figure's pose, before detailing the rest of the figure, see Fig. 2. The concept of creating a skeleton serves as a powerful strategy for visualisation design. Different paths can be imagined, from x-axis, circles, zigzags, to space-filling curves such as Hilbert, Moore or Peano curves.

Our conceptual idea is straightforward yet impactful: establish a skeleton, represented by a flowgraph, to provide the structure for the visualisation. Encode the parameters of the visualisation design using a genetic metaphor, where various components such as symbols, colours, and textures represent genes. Swap in and out different aspects of this 'gene', such as symbols, colour, textures, to generate diverse design variations. The flowpath acts as a backbone to the visualisation, on which we constrain the path through an envelope description, which confines the scope of the visuals. We map objects along the path in sequence, and map data values on visual attributes of the objects to

encode data. The power with this strategy is to explore designs quickly by exchanging values in the gene. Effects and filters can be mapped along the path, such to merge or intersect overlapping objects, or accumulating transparency of overlapping objects, or morphing them together in some way. Figure 3 outlines this concept, exemplified by a simple bar chart on an angle. The flowpath is defined by a contiguous set of vertices v_0 to v_{n-1}, but does not need to be continuous in space, as it can contain spatial jumps, which allows visual separate designs (such as small-multiples) to be created. In the case of the bar chart, rectangular objects placed between the vertices, which are scaled and coloured by the data. Other visualisation forms are made by changing the sequence order, locations of points, form of objects, envelop, alignment of the actual object (middle, top, bottom etc.), and applied filters on the placed objects.

While the idea is simple, the challenge is to extend this principle and to develop a framework around this model, that allows users to express their creativity and to be able to easily make their own visualisation designs. This work substantially extends our poster presentation at IEEE VIS 2018 [32] and workshop paper [31]. We make four contributions: **(1) A path-based model** for the creation of visualisation designs, visualisations are structured on this backbone skeleton path, Sec. 4. We present the model such that other people can implement and expand it. **(2) The Genii system** implementation contains a component library, renderer and builder application, enabling users to create visualisations by constructing genes through the path-based design model. Outputs from the builder are displayed in Figs. 1, 12 and 13 and Sec. 5. Additionally, we showcase a mobile watch implementation that how the system can create visualisations for a smartwatch Fig. 11. **(3) Several case studies**, where we re-create published visualisations using our path-based model Sec. 6 and Fig. 12, and **(4) an evaluation** of the tool and model, Sec. 7.
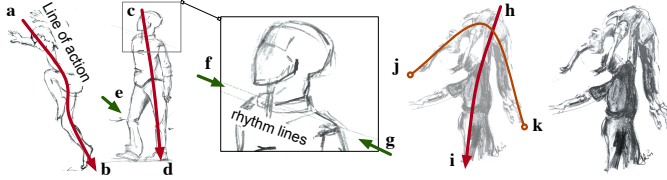

Fig. 2: Reilly lines of action [50]. The method is a constructive approach that emphasises primary lines of action, a→b, c→d, h→i, j→k, and secondary forms (rhythm lines), e→, and f→g.

2 BACKGROUND

We have long advocated the use of alternative visualisations for various purposes. In the design phase, it enables individuals to explore various potential solutions [73]. In exploratory visualisation, different forms can focus on specific tasks or facilitate particular interactions [70]. Meanwhile, in explanatory visualisation, different views can help individuals understand the information more effectively [71]. Several ideas converged to inspire this work, including alternative design, the skeleton pathway, and design similarity.

Our classes on creative visualisation design employ the Five Design-Sheets (FdS) approach [73], where multiple sketches are an essential part of the initial sheet. A key principle of this creative design process is encouraging individuals to produce numerous **alternative** rapidly drawn sketches during the design planning phase. This method allows for a more effective exploration of a wide range of potential solutions. In our sketching approach, we emphasise the importance of drawing confident yet simple lines. These initial lines serve as the foundational framework, akin to a **skeleton**, upon which more detailed and refined elements can be built. This method helps designers establish the basic structure and proportions of their work early on, providing a clear guide for subsequent stages of the design process. By focusing on strong, assured strokes, individuals can better visualise the overall composition and make necessary adjustments before committing to more intricate details. In another course, we provide general sketching

skills with an emphasis on drawing human figures. In art education there are several approaches. Bridgman [14] emphasises the masses of the body and their connections. Loomis [46] works with proportions and builds up the body from simple shapes — there are similarities between ideas of sketching by Loomis [46] and scaffold shapes, with the diatoms approach by Brehemer et al. [11]. We however focus on Frank Reilly's method [23, 50], which guides students to draw simple lines (flow lines of action) along with secondary structures (rhythm lines), see Fig. 2. Artists are encouraged to first sketch the line-of-action before adding in the body, legs, head and other forms. It is a constructive approach. By first defining the lines of action the human shapes are more expressive, and the sketch is more convincing. Simple shapes can be defined by single lines, e.g, a line top to bottom when someone is standing Fig. 2(c→d) or an S-shape when someone is leaping (a→b). Complex shapes can be expressed with multiple lines of action as (h→i) and (j→k). As we examine these sketches, our attention is instinctively drawn to the principal lines of action. We imagined building visualisations by similar methods.

Our third source of inspiration, **similarity**, and design modification. This goal emerged during consultations with a company specialising in nursing applications. They aimed to create alternative visualisations for presenting patient health data, ensuring consistency in visual representations for patients with similar ailments while being distinct enough to differentiate between individual types of ailments. The approach of similarity and design modification offers many benefits. Modifications allow designers to experiment with variations without losing the core essence of the design, balancing innovation with coherence. This method enables the creation and evolution of a suite of ideas, allowing people to explore and refine concepts effectively.
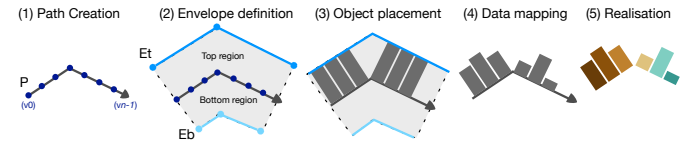

Fig. 3: To create the visualisation, first define the sequence, describe the envelope, place objects, map data and add any final filters to realise the visualisation. We represents the points P as a sequence of vertices v_0 to v_{n-1} (1). This acts as a skeleton that forms the supporting structure of a visualisation. The envelope (2) acts to control the scope of the placed objects (E_t to E_b). Data is mapped along the sequence of points (4), where in this example data is applied to size the objects to create an angled bar chart. Finally colour is mapped to the sequence data to complete the visual design (5).

2.1 Design goals

We define six design goals that define specific objectives and intentions to guide the creation and implementation of our path-based design model for alternative visualisations.

Our first goal is to develop a method to **compose** the visualisation from a series of straightforward steps. This is beneficial because it enables a module design, where individual components can be developed, tested and maintained separately, and it promotes re-usability and scalability. We explored the use of higher-order functions, like such as *map*, from declarative and functional languages. *Map* is used to apply a user-defined *function* onto each element of a *list*. Where for instance, `const l = [1, 2, 3]` makes a list `l`, and `const r = l.map((v, k) => v * k)` multiplies down the list, and the result `r` becomes a list `[0, 2, 6]`. We conceptualised that a *path* is a sequence of coordinates, where developers would apply visual design elements onto every facet of the path. Data is mapped to each object along the path. The visual components are combined to form a sophisticated visual design.

We require mechanisms to enable **paths** of different shapes, and allow them to be continuous and non-contiguous. This is beneficial to allow flexibility in data presentation, varied visual interpretations, and support different visualisation types. One example use of paths, is from the data-sonification paths of Roberts and Franklin [72]. They use a path to scan the visualisation, map a kernel each position, which is turned to

sound. Another example is the PATH definition in the SVG web format, which can describe non-contiguous paths `<path d="M0 0 L0 100 L 100 100 Z M 200 0 L 100 100 L 200 100 Z" />`. Consequently, linear paths would make a bar chart, whereas circular paths could be used to create a dial visualisation.

Secondly, we recognised the importance of having self-contained visualisations that render within a **viewport**. The visualisation can encapsulate all necessary components (data, styling, interactions) within itself, makes the system portable, can be optimised for an available screen real estate, and offers portability as it can adapt for different viewport sizes. For instance, the nursing application visualisations would need to be sized appropriately to appear at the top of other text descriptions of the patient. The visualisation for our smartwatch (977 mm^2 for an Apple Watch 5, 44mm, or a few centimetres in physical dimensions) represents a small viewport. Thus a design goal is to allow a range of paths to be created, and constrain them to a viewport. Our approach is to clip the visualisations using an envelope (see Sec. 4.1).

Third, we require a **deterministic** system. A deterministic system ensures that given the same input or conditions, the visualisation will always produce the same output. It means the system is consistent, easier to debug and test because the behaviour is predictable, and results can be compared. This is crucial for scientific research, where reproducible of results is essential for validation and verification. While random paths could yield interesting and unique designs, they would vary potentially with each run. Our solution is to use a seed. Such seeds are often used by game developers, to generate the landscape and keep it consistent between sessions. Our solution was to save the design as *gene* expression that can be used to define the whole complex visualisation, and to enclose the seed in the gene expression.

Our fourth requirement is to enable designs to be **similar, yet different**. This helps to develop a range of designs, help to maintain brand identity across ideas and maintain a recognisable style and personalisation. For instance, using graphical elements to represent a collective group (e.g., a ward of patients) while ensuring each individual patient stands out visually. Or to enable users to create a variety of visualisation designs that share common traits yet are distinct and visibly unique. It should be adaptable and extensible, capable of designing both connected and non-connected visualisation types across a wide range, and apply to each part: object or colour palettes, paths, data and so on. We achieve this by our modular approach of using a *path*, *envelope*, *object placement* and *filters* and storing it in a *gene*.

Fifthly, in addition to being deterministic, we require the visualisation generation algorithms to be suitable for real-time use and **computable**. Whether heuristic or exact decisions are made, we needed to be sure that we could compute and finish the design. The design constraints that we impose on our model are not as strict as, for example, with graph drawing [4] or label placement algorithms (e.g., [79]). But there may be ways to make the model render depend on its context. Much like L-systems or context-grammars. Indeed our goal is to foster the creation of creative outputs, contrary to the goals of a developer who, when it comes to graph layout and label placement, is typically concerned with readability. In our case it may not matter if the objects overlap, and in fact, it may result in a nicer visual if they do overlap. In addition, we do need the algorithm to complete in real-time, and there may be situations where a heuristic is used to make a choice over two positions. We wish to avoid situations whereby the algorithm merely alternates between two choices and does not terminate. This is more an implementation challenge, and our solution is to focus on greedy approaches, such as to choose the first/ naive choice, or to follow a worsening choice strategy, like the Tabu search [94] algorithm. Finally, when such decisions are made, we need to save the seed or choice, such that the design is still deterministic.

Sixth, we require an **extensible** model, so that different path types, or layout algorithms may be used in the future. Our solution is to divide the model into modules: path, envelope, object placement and effect filters. We believe that the path model is implicitly extensible, because of its composable nature; it would be easily possible to change the path to 3D, or add different elements onto the path, and to merge, clip or perform other functional operations to extend our basic ideas.

3 RELATED WORK

Inspired by the structures of Reilly [50], we define a flowpath (line of action) and envelops (rhythms) to define a skeleton on which visual shapes can then be added to create a visualisation. This pathway represents the primary **visual flow** of the visualisation. These observations align closely with theories of visual flow [16, 59], which have been substantiated by eye-tracking studies [56]. We further discuss visual flow in the design goals in Sec. 2.1 when we expand on our goals, and in Sec. 4 when we talk about paths. Ware [90] discusses a similar concept to reading a chart, often referred to as "eyeballing the graph". This concept emphasises the intuitive process by which individuals visually interpret graphical data, such as scanning a chart from left to right along the x-axis. Conceptually, this flow provides a convenient sequence, which we use as a backbone to create a visual design. Bertin's classification of image space usage [6] uses icons to show the specific structures (linear, circular and so on) following also this idea of flow.

Creating glyphs [9] aligns with some of our goals as they provide a method to present complex data effectively, are scalable, and researchers aim to create variations that are both similar and distinctive. We direct the reader to several relevant surveys. Ward [89] investigates glyph placement strategies. Borgo et al. [9] reviews design guidelines, techniques and algorithms for glyph generation. Ropinski, Oeltze and Preim [74] survey the use of glyph visualisations for spatial multivariate data. Fuchs et al. [26] review 64 user studies of data glyphs, provide a meta analysis of the results and look into related design trade-offs. There are many ways to design glyphs (see Borgo et al. [9]). For instance, Legg et al. [41] uses a quasi-Hamming distance to ensure sufficient individuality in the resulting glyphs. Ebert et al. [21] explore procedural generation, based on the automatic generation of glyph shapes using superquadric functions (also see Patel and Laidlow [60] and Gerrits et al. [27]), fractal surface displacement, and implicit surfaces. Another strategy is metaphoric; to either take an object (such as a pea pod, as used by Koc et al [39]) or analyse an image to generate the component structures, that can be built into the final glyph (as used in MetaGlyph [95] or glyph from icon [65]). An alternative approach is to evolve generations (e.g., [47]) or use learning systems. Indeed, glyph learning systems is a dynamic and promising research area [44, 92]. A further strategy is to have an underpinning structure, to base the glyph on. For instance, Ying et al. [96] in their circular glyph maker, define interior, intermediate and exterior structures. Brehmer et al. [11] use a generative approach, mixing a scaffold of a shape (rectangle, triangle, etc.), encoding channel and mark shape palette; similarly, Khawatmi et al. [37] start off with simple geometric shapes. While Keck and Engeln [36] use a star. We, however, use a different structure. We use a flowpath as the backbone of the design. Our paths can be created manually, randomly, procedurally or from data (see Sec. 4). While Pereira et al. [62] do use a simple skeleton line, their purpose is to create fonts, not glyphs or visualisations.

Glyphs have been used in a wide range of situations. For example, Kindlmann and Westin explore glyph packing in medical visualisation [38], whereas Maguire et al. [48] propose a taxonomy-based glyph design approach, intended for visualising experimental design workflows encountered in biology. Legg et al. [40] explore their use in real-time sports analysis. Pearlman, Rheingans and Des Jardins [61] focus on glyph-based multi-variate visualisation in different scenarios, such a analysing student applicant pools, network traffic and fantasy football team building. Kammer et al. [35] create a small-multiple zoomable dashboard of glyphs; in a similar grid-based approach Rees et al. [66] present hierarchical glyphs, to overcome the overplotting challenge when placing glyphs, and Pires et al. [64] a treemap of glyphs. Suschnigg et al. [82] use glyphs to present anomalies in time series data. Glyphs are also used in more unconventional depictions, such as haptic glyphs [72], or in mobile Augmented Reality environments (e.g., Chen et al. [98]), or hand crafted for scientific visualisation [97]. And have been used to represent uncertainty [24, 27]. Many researchers have investigated the effectiveness of glyphs. For example, Maguire et al. [49] investigate how motifs can increase comprehension of workflow visualisations. While Hu et al. [30] analyse statistical orderings

of star glyphs. Other researchers have evaluated specific glyphs, such as whether Chernoff faces [17] are readable [55], the effectiveness of temporal glyph designs [25], glyph placement strategies on maps [53], analyses of historic use of music glyphs [57], effectiveness of flower-versus star glyphs [86], and how some designs are more glanceable than others [7].

Creating small visualisations aligns with some of our goals including rendering in a viewport, scalable, and quick to create. Tufte's [85] sparklines are small, simple, word-sized graphics with high resolution. Saito et al. [75] use two-tone pseudo colours for quick data overviews, Willet et al. [91] developed scented widgets embedding visualisations into UI elements, Perin et al. [63] created sportlines for player movement in soccer, and Goffin et al. [28] designed word-scale visualisations with flexible placement. Brandes et al. [10] introduced Gestaltlines, multivariate sparklines using Gestalt principles. For a review of word-sized visualisations, see Beck and Weiskopf [5].

Many developers have created tools to assist in visualisation creation. Early Modular Visualisation Environments (MVEs) like AVS, IBM DX, IRIS Explorer or Visage [77], enabled users to craft custom solutions by connecting various modules together. Visual authoring systems like iVisDesigner [67], Lyra [76], and DataIllustrator [45] assist designers in creating custom visualisations. While effective for bespoke designs, our focus is on rapid design and the generation of multiple diverse visual depictions. Several tools assist in hierarchy visualisations: Visception [80] focuses on nesting, HiVe [81] uses a grammar for hierarchical layouts, and GoTreeScape [42] employs a declarative grammar. Preset-based visualisations [78] and StructGraphics [83] enable designing visual structures before applying data. Structured mock-ups rely on display space partitioning [87]. Li et al. [43] identify features of visualisation designs that they explain as being similar to rRNA genes, that they use to build a high-dimensional phylogenetic tree. Influential studies include Brehmer et al.'s mobile time range visualisations [13] and Blascheck et al.'s smartwatch glanceable visualisations [7], which we replicate using our path model (see case studies, Sec. 6). Our method is similar to TimeSplines [58], but allows path creation and attribute swapping with a drag-and-drop interface.
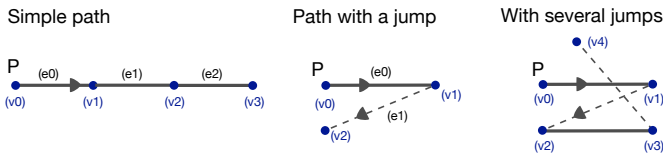

Fig. 4: The path is made from a set of vertices. Objects encode the data and are mapped along the path.

4 THE PATH-BASED MODEL

Referring to the simple example Fig. 3, we define our path P to be a directed path that is made from a finite series of vertices (see Fig. 4). While typically we equate vertices with (x,y) locations, it is possible to imagine that they could be mapped to other modalities; e.g., with sonification the sequence (path) could be notes on a music scale, sounds over time. A walk along this path therefore moves alternating between vertices and edges, $v_0, e_0, v_1, e_1, v_2, \ldots, v_{k-1}, e_{k-1}, v_k$. Our path is a *flowgraph*, in that we utilise every point in the path, and provide an order on the set of points from the first (v_0) to the last (v_k), which define the entrance and exit points, respectively. The path acts as a skeleton, on which we place visual marks. The path is contiguous, where there is a neighbourly structure from one vertex to another, however it does not need to be continuous, because we allow edges to jump. The path can also intersect and cross over itself, and can return to a previous position. However, this is achieved by adding the same points to the the path. The path itself does not hold any logic, it is merely a route to direct how the geometry is placed. Logical operations take place when the objects are placed along the path.

The flowgraph is used to express a *visual flow* through a visualisation design space [22]. The way someone 'eyeballs' a chart can be influ-

enced by the cues surrounding it. The arrow of an x-axis and the nature that it is a line, draws the eye from the left side to the right. Visual designers can change how people look at certain parts of the picture; they provide visual cues to lead the viewer to *flow* in a particular *visual direction* [2]. For instance, gradient lines direct the viewer along the direction of the lines, colour gradually getting darker would encourage the user to look in that direction, whereas size changes would encourage attention, while visual cues such as arrows would focus the observer to focus on the arrow head or whatever the arrow points to. The way we look at the visualisation is not random. We direct our eyes to interesting features, and follow the story of the graphic.

Visual flow is evidenced in many activities that involve a human looking through their eyes. For instance, a person reading an English text would read from the top-left of a page along a line to the right, and from the top of the page to the bottom (in a **Z-pattern**). Or someone looking at a webpage would glance at the top left and move down the page to the right. This dominance is named the **Gutenberg pattern** [59]. The visual flow, allows users to eyeball the visualisation [84] and permit the user to notice patterns such as heights of graphs, trends and outliers. Consequently, it is possible to analyse the visual flow and the dominant flow from the composition of the visualisation. In fact eye tracking can help to reveal these patterns of user behaviour, demonstrating for instance that web-users fixate on webpages in an **F-pattern** [56]. The F-pattern also suggests that most users will not read to the end of the row, but they skim down the page reading quickly and scanning the text. Eye tracking has been used widely in visualisation, for node-link diagrams, trees, and to understand how a user searches, perceives and explores a visualisation [16]. Matzen et al. [51] explain that visualisations are read differently depending on the task, and the type of visualisation, with unfamiliar visualisations inciting more eye tracked fixations. From these fixations they have developed salience metrics to predict how users will view visualisations [52]. But even before eye-tracking studies, Bertin [6] classified several arrangements of how to utilise the image space, and provided a visual naming scheme based on these methods, including the dimension of the plane ⟶, repeated categories ⇢, collection of items ᵐ⟶, cumulative quantities on a dimension ⊢⟶, circular ↻, irregular ⟋, and regular arrangement ⟍. These display models are summarised by Roberts [69]. Brehmer et al. [12] extend Bertin's scheme which they use to classify storytelling and timeline techniques. There are similarities in their design space to ours; while they focus on specific instances, we have developed a path model that can be used to create different visualisations, and our gene allows us to easily change between several design strategies. Additionally other path types are useful for our designs, including spiral ↻ and golden spiral ⟲. By defining the vertices of the path we can recreate each of these styles.

Our path is piecewise linear. We treat each of the edges as discrete parts. This is a sensible solution, because (i) the data that we map onto the visualisations will always be discrete in nature, (ii) it is clear how to map data onto the path, because it can be easily mapped along the edges, and (iii) even if the original data is continuous in nature, a developer we would have stored a discrete (sampled) version of it in a database. This strategy does not inhibit continuous or smooth-looking visualisations; on the contrary we control such smoothing at the envelope (Sec. 4.1) and object placement stages (Sec. 4.2). We note that it would be possible for us to have defined a curved path, such as the cubic beziers definition in the SVG PATH. Yet, one disadvantage with this solution is that while such a path would be continuous it would be unclear how to map the data onto it.

Finally, it would be possible to create unsuitable paths, such as those that are unnecessarily complex, contain too many vertices that may take a long time to render, or paths with the points that are all located in a small space that produce overplotted images. However, our goal is not to constrain potential creativity, and hence we do not constrain the length or position of the vertices in the path. Nevertheless, in our implementation we make checks on the path, e.g., eliminate null vertices, that the smallest object is larger than a pixel, and clipped to render in a unit space.
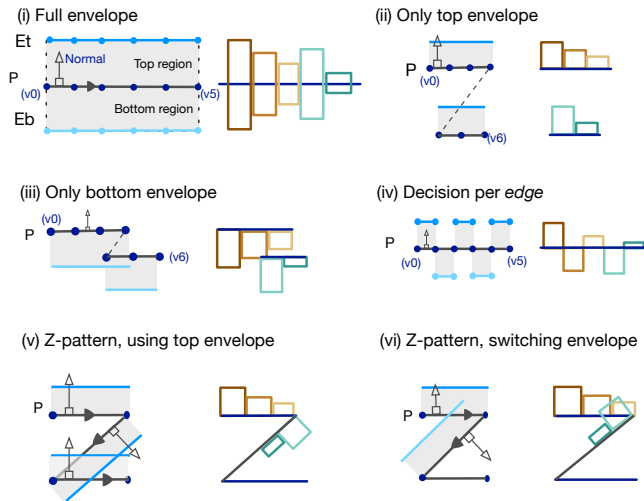
Fig. 5: It is possible to control the placement of the objects by changing parameters of path and envelope. (i) Shows centred objects along the path; (ii) uses only the top envelope and snapping the objects to the path, (iii) puts objects under the path, (iv) can alternate between top and bottom envelop, (v) z-pattern showing how the return angle places objects underneath, (vi) and by switching the envelope on the angle objects can be placed above the path.

4.1 Envelope

The *envelope* is defined by a finite set of vertices of a top (E_t) and bottom (E_b) path, and sits on the main path (P). The envelope has a Normal (N) that defines a vector perpendicular to the path P, which is used to determine the top and bottom envelope regions. While the *path* describes the visual flow and positioning of the elements, the *envelope* determines how the objects appear along the path, and defines the baseline of where the objects are placed. For example, the objects can be placed *centred*, *above* or *below* the line through controlling which part of the envelope is used, as shown in Fig. 5. The envelope defines a range, which is the viewing region where the graphical objects will be drawn. It can be used to clip the visualisation object should it be larger than the range of the envelope, or filters can be applied (during the object placement stage) such to smooth or filter and merge objects together, see Fig. 8 and Sec. 4.2.
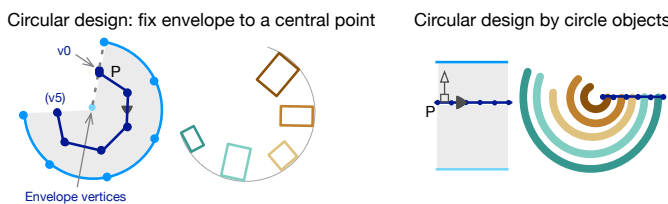

Fig. 6: Different circular visualisations can be created through circular paths, and by setting the envelope points to a fixed location. Or by placing circular objects.

In many situations it is easy to conceive that the envelope E_t and E_b will be parallel to the path. The envelope is defined by a series of vertices (much like the path), and progresses along with the path to define the visual flow of the visualisation. Fig. 5 shows some example path and envelope combinations. Fig. 5(i) demonstrates the full envelope, with a parallel path, which is used to centre the objects as they are placed along the path. Not only can the path be non-continuous, but also we can determine if the objects sit above or below the path, by changing the envelope, as shown in Fig. 5(ii) and (iii). In fact, we can use this technique to create a small-multiples display. Fig. 5(iii) also shows that it is possible to create overlapping objects. We can switch the choice of the envelope, such to move the baseline of the object from top to bottom, as shown in Fig. 5(iv). Finally, Fig. 5(v) and (vi)

show zig-zag paths, where (v) is the normal situation whereby the path naturally turns the objects upside down, as it goes through the diagonal part of the Z-path. This example is important, because a designer may wish to keep the objects upright. This is achieved by switching between top and bottom envelope, when the path turns.

By changing how the path is constructed and how the envelope and path work together, and how the objects are placed we can describe different structures. In particular, the envelope does not necessarily have to follow the path, so for instance, by fixing all the vertices of one envelope (E_b in this case), we can create a circular visualisation (Fig. 6). An alternative circular visualisation design can be created using a linear path, and placing circular objects.

4.2 Object Placement

Once the *path* and *envelope* have been defined, we now decide what *objects* to place down, and how to apply them. These objects are the geometrical shapes that become the visual marks that encode the data. We have used many shapes including circles, arcs, donuts, triangles, rectangles and lines. The location of the objects is controlled by the path and envelope, but the visual appearance and any effects are applied in this step. Objects are usually placed in path order and overlapped, where subsequent objects will be placed on top of previous ones. Therefore, if a different order is required, either the path vertices need to be re-ordered, or the data order changed. When the objects are placed they are positioned, scaled, rotated, and any filters applied, before applying the clipping of the path, or other constraints applied by the path and envelope. For example, a stream graph visualisation can be created by defining a path (P) drawing one outline, using the same vertices to draw the second outline, subtracting the objects, and smoothing it with a filter (see Fig. 7).
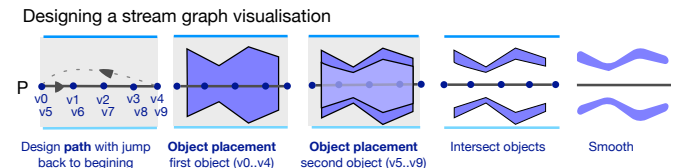

Fig. 7: Building up a stream graph visualisation, through several coincident paths.

By placing different objects and applying a variety of filtering operations we are able to create typical visualisations, as well as new designs. There are many filters that could be applied, including smoothing, intersection, union, blend, blur and shadow. Overlapping objects allow several effects to be applied. Fig. 8 shows three scenarios: (i) objects placed on order, with later objects being positioned on top, (ii) later objects cutout sections, and (iii) combining objects with a metaball operation [8].

We can define more complex visualisations using more abstract paths, see Figs. 9 and 10. For example, to create a scatterplot (Fig. 9) either the path is defined as a straight line across the x axis, where the vertices are determined by each of the x data values, and the objects being placed are the scatterplot circle symbols. Or an abstract path is defined, that joins every datapoint together, and is positioned such that each data point sits on the *edge* between two vertices. The scatterplot is drawn by moving along this abstract path. This second approach would be helpful if considering label placement. Other paths can be implemented,

4.3 Data mapping

The data mapping stage of our model offers some interesting choices and opportunities. We can map *data* to any part of the model, such as the *path*, *envelope*, object, object position and its appearance, and apply filter operations on the objects (based on data). For example, a simple bar chart is created by defining a regular set of path points (v_0 to v_n), choosing an envelope that sits on the path, placing rectangles such that they sit on the path, and mapping each data value to adapt the height

Different filtering operations

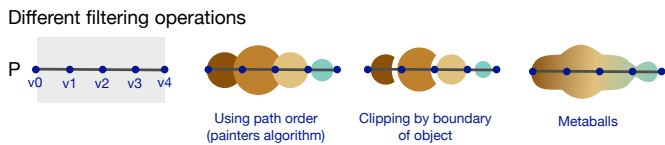Fig. 8: Illustration of an envelope overlap and three possible strategies for dealing with it: 1 objects are placed in order, with the next object overlapping the previous, 2: The previous object is cutout, 2: The objects are combined using a function (in this case Metaballs).
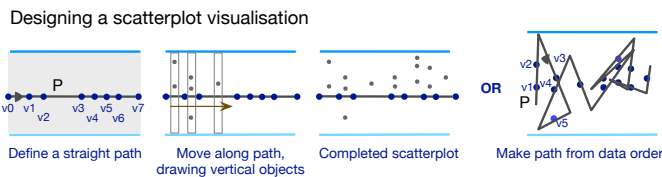
Designing a scatterplot visualisation

Fig. 9: Two strategies for representing a scatterplot using the path model. 1) The path is defined across the design space and scatterplot marks are placed vertically. 2) A path is constructed from the data in order and scatterplot marks are placed along that path.
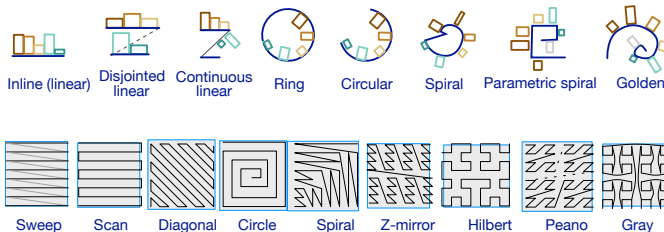

Fig. 10: Example path types. Lower are space-filling path types, including sweep, scan, diagonal and Hilbert, Z-mirror, Peano, and Gray [19]. Each space-filling path traverses every grid point, in a continuous, non-overlapping manner. Hilbert, Peano, Z and Gray look to place nearby points in the grid such they become close together in the curve.

of the rectangle. This will create a black set of rectangles (the default colour), so to add different hues we can map a colour to the fill of each rectangle. We could adapt this further, for example by mapping data to vertex position we can change the width of each rectangle. Same data values could be simultaneously mapped to different parts of the model, or data to different parts to realise a multivariate visualisation.

Data mappings can be dependent on previous decisions. This is possible because we have a precise ordering to our model (path creation, envelope, object placement and mapping). For instance, to create stacked bar charts, subsequent rectangles start where the previous ends. Trees and other hierarchical structures can be similarly created. While we do not allow conditional logic to be embedded in the path, we do allow the path to jump, therefore simple tree structure can be created by placing an object, and then jumping the path to the start of that object. While the path does provide a skeleton, and describe the general visual flow, (in the most extreme cases) the geometric objects can move away from the physical constraints of the path. For instance, a rectangle can be positioned on an edge, with its exact location being jittered (or offset from a data value). However, when such operations are applied, it is important to consider the requirements of being composable, self-contained and deterministic. Any seed value (for stochastic jittering) or data values for relative positioning, must be encoded into the gene expression such make certain the visualisation is deterministic.

5 IMPLEMENTING THE PATH MODEL

The Genii builder is shown in Fig. 1, resultant output in Fig. 13, and case studies in Fig. 12. The system consists of a component library, renderer and builder application, see Supplemental Materials. We first made sketches, which helped us to refine the initial concepts and the model. We implemented an initial prototype with JavaScript and HTML5 Canvas [32]. This prototype included features for object unions, intersections, and metaballs [8]. However, we encountered difficulties in creating paths that depended on the positions of other objects. We needed a more modular approach to the implementation. We chose React.js because it is modular in approach, uses HTML5, CSS and JavaScript, and in particular we are able to pass detailed data to control the DOM. It also provides rapid prototyping and multi-platform support. It also allowed us to develop the full browser application and migrate it to the Apple watch, which links with a iPhone, see Fig. 11. We detail the implementation below:

Flowgraph. There are three ways to implement a path in Genii. First we define set paths. These include: *inline linear* which provides a uniform straight line distribution of points horizontally across the canvas. *Disjoint inline* a path with jumps. *Ring* which is a circle of points, with

the same first and last coordinates. *parametric spiral* which is formed by a series of straight line paths with 90° rotations and increasing path lengths. Hilbert space filling curve that passes through every point within a square grid in an uninterrupted, non-overlapping fashion. Each path is adaptable, for instance, by anchoring one reference point to the center of a circular path, we can generate a star plot. Second, we provide a path editor for user-generated paths. Individuals can visually place points, and Genii records the path accordingly. Finally, paths can be added to the code, by replicating and editing the React component.

We implement the **envelope** in Genii by clipping. Elements are placed (centred aligned, top or bottom aligned) along the path, based on the Normal. Which can be used to create objects that alternate between above, below or on the path. **Object placement** is achieved by placing geometric shapes between vertices of the path. As shown in Fig. 9, the distance between vertices determines the default location of the object placement. Then the data values will then adapt this position. For example, one chosen shape can be mapped along the path, and its location adapted by the data. Or the shape can alternate its placement (sitting on the line, or below it) dependent on its position. Or different shapes can be applied, to each location along the path, dependent on the data. Genii includes several basic geometric shapes, including: *lines* that can be used to create triangles, or outline shapes; *triangles* by drawing a line between the vertices and the central point is adjusted by the data value; *arcs* using the central as a Bézier control point of a quadratic curve, rectangles; *circles* and *ellipses*, where the radius can be controlled by the distance between the vertices, or by a constant, or by a data value. We have implemented several filter operations, including object/primitive fill, linear gradient fill and radial fill, change of object stroke, and metaball merge.

We needed a way to load **data** and define the sizes of the visualisation. We store the data in a JSON object that includes the name (a string), data value (float) and data range(float) of the data for each category, the *width* and *height* of the visualisation (in centimetres), and padding around the edge of the visualisation. This provides a convenient way for us to save the data created in Genii. We store the visualisation design in our *gene* expression. Users hover over the top right hand side of the Genes panel (see Fig. 1 middle). The saved file contains a set of enumerated types stored as parameters in
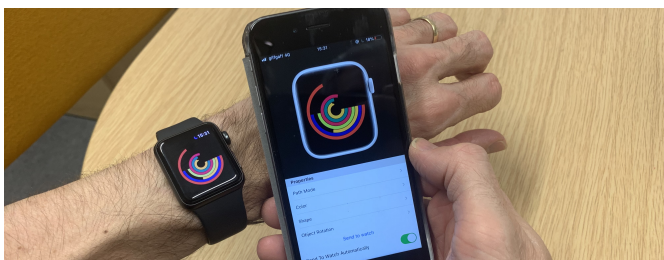

Fig. 11: Photograph of the Genii iOS application where people can build the visualisations and load them to the companion smart watch.

JSON. Our application will only save valid genes, but should they be invalid (edited by an external user) our application first validates the gene expression and warns the user where the problem lies, this is achieved using the transpiler (Babel.js). The gene saves many properties (see Listing 1), including: Path Mode, Path Rotation, Path Point Distance, Path points, Object Shape, Object Colour, Path Grouping. To make the application *deterministic*, the user is asked to *name* their visualisation, which is hashed to a 32bit integer value. We use a hashCode() function of string s to create the product sum of the string's character codes, as follows $h(s) = \sum_{i=0}^{n-1} s[i] \cdot 31^{n-1-i}$.

Listing 1: Example gene; used to save the path-based visualisation

```
1   const params = {
2       shape: Gene.shape.BAR,
3       color: Gene.color.FROM_DATA,
4       color_key: Gene.color_key.OFF,
5       path_points: Gene.path_points.EVEN,
6       path_mode: Gene.path_mode.INLINE,
7       path_rotation: Gene.path_rotation.NONE,
8       path_grouping: Gene.path_grouping.NONE,
9       object_rotation: Gene.object_rotation.NONE,
10      object_size: Gene.object_size.FULL,
11      filter: Gene.filter.OFF,
12      debugging: Gene.debugging.OFF
13  }
14  const Gene = new Gene(params);
```

6 CASE STUDIES

To demonstrate the model we recreate several visualisations from papers (Blascheck et al. [7] and Brehmer et al. [13]) and an imaginative design from a concept sketch. Each of the visualisations from the papers are small visualisations that would be suitable for mobile devices. For each, we present their visualisation and analyse the structure, show a copy of their original visualisation and explain how we recreate them in our path model.

To recreate the bar chart, donut chart and radial bar chart of Blascheck et al. [7] we need to imagine the visual path. The left column of Fig. 12 (top) shows their original visualisations, our paths and our recreation, see Fig. 12). Understanding the visual path of the bar chart is easy, where a straight path would readily recreate this visualisation. Similar to the examples in Fig. 5 we recreate this chart using a straight path, envelope that stretches to the top of the design space, rectangles aligned on the path encoded with data. The donut chart can be created in several ways. We could treat this as one circular object that is placed on a simple path (see Fig. 6) or a circular path. Circular paths can differ, depending on whether the objects would sit at the top of the envelope or the objects sit on the path. The best decision depends on how the donut will be used, and how it displays data. Because the width of the donut's ring is not defined in the original paper we decided to create the donut chart with a circular path, truncated circular segments of a cone (the objects) sitting on the path. There are different ways to implement the radial bar chart (as discussed before, see Sec. 4.1 and Fig. 6); we chose to implement it using a straight path with circular objects, encoding the data to fill a percentage of the full 360° arc. For each visualisation the colours follow a colourblind friendly differentiator scheme between data visualisation elements.

We recreate two visualisations from Brehmer et al. [13], shown in Fig. 12, which can be done in different ways. For the first visualisation we use a straight path; but on the outset it may not be clear to ascertain how many paths are required. Either the *flow* is considered as one path and we place a complex object at each point; or it can be appraised as several paths, that are located at the same point (like the streamgraph example, Fig. 7). While the outcome would be the same, differences exist in the way to build the path and apply the data. We use a straight path with envelope above it. There are two other possible path and envelope combinations: i) path through the middle with envelope above and below; ii) path along top with envelope below. Alternative i) requires extra checks on the data and a more complex placement calculation. The differences between alternative ii) and the
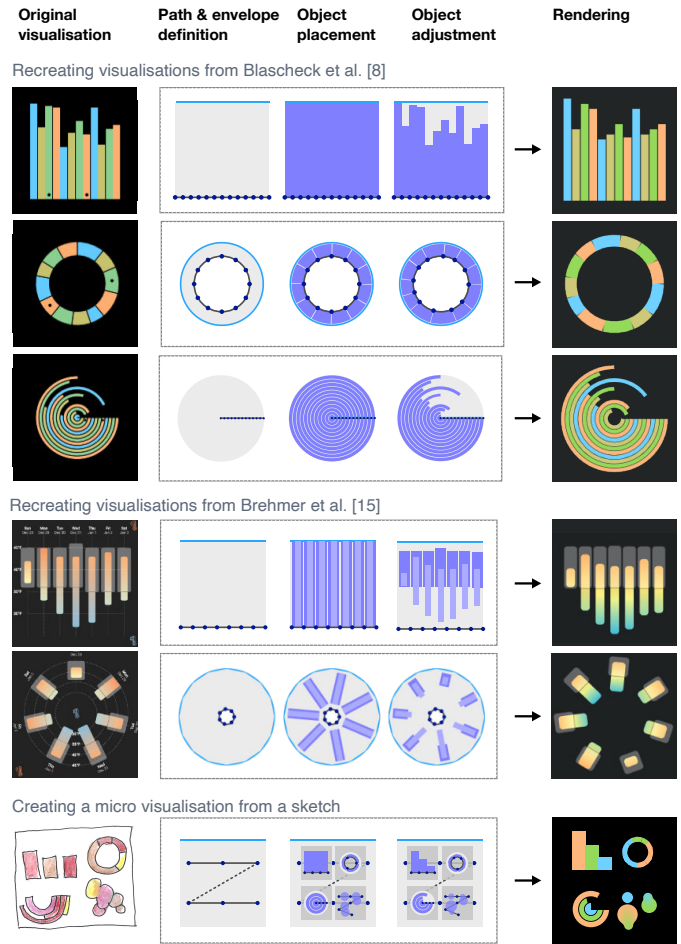

Fig. 12: Recreating visualisations by Blascheck et al. [7] (top) and Brehmer et al. [13] (middle). Also creating a visualisation using a non linear path from a sketch. Showing the original visualisation, an explanatory depiction of the path used, and the final rendering (right).

chosen path are subjective and we make the decision to place it at the bottom was made to match with an *y* axis. In addition we map colour. Three visual variables are used to describe the data: a) colour is a linear gradient from blue to yellow to orange for daily temperature range, and an opaque white colour to denote the average temperature range; b) the size of the element shows the size of the range of the data point; c) the placement of the element shows the start of the range. We map a filter to the objects, to round the corners (in the same stylistic way), and we do not include labels (but could do so by adding a new path, and placing text labels). We create the circular visualisation using a similar path to the circle chart of Brehmer et al. [13], but apply different shapes and align the objects to the outer part of the envelope.

Finally, we create a disjointed design which has four separate visualisations (see Fig. 12). We sketched the idea and then created a disjointed path. We start by defining z-pattern path, we insert extra vertices into each part of the z shape, such that we can describe the individual visualisations. For the first three (bar, donut and circular chart) we followed the descriptions as described above. The final visualisation uses a random path, and place circles along the path, applying a metaball filter to blend the circles.

7 BROWSER AND MOBILE APP USABILITY EVALUATION

Our emphasis lies in crafting diverse designs rather than assessing the effectiveness of the resulting visualisations. For evaluations of their efficacy, we direct readers to the work of other researchers in the field (e.g., [7, 13]). To evaluate the usability of our path model, we used our desktop implementation (see Figs. 1 and 13) and mobile phone/watch tool (see Fig. 11), with default paths available. Users were given a
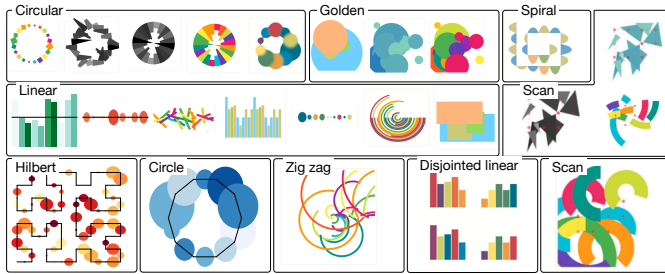
Fig. 13: Genii visualisation designs, showing a range of different flow paths.
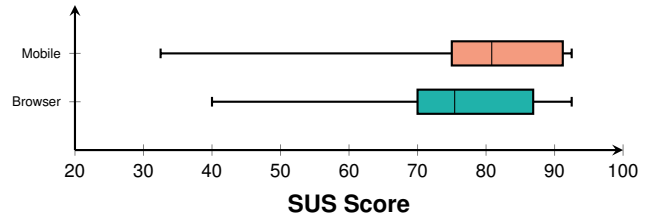

Fig. 14: Box plots of SUS scores. The usability of each tool was good, and therefore should not affect adversely the generation of each visualisation.

five minute introduction video showing how to use the tool, followed by five minutes self-guided exploration, where they could create their own visualisation designs. They identified if they "liked" or "disliked" each creation during the desktop study by clicking on the images in the gallery. On mobile, users had the ability to send the visualisation to the Apple Watch if they liked the design. We chose a short design duration, because we wanted users to rapidly create many visual designs. After five minutes they completed the System Usability Scale (SUS) [15] questionnaire. Participants also reflected on their design process, by answering (1) "Describe how you would like to use these visualisations in day to day life"; (2) "Describe some aspects that you liked about the visualisation design process"; and (3) "Describe some aspects that you disliked about the visualisation design process".

Participants. An initial pilot study was carried out using four participants (two web developers, one psychology student and a student nurse) recruited over social media. We used our JavaScript evaluation framework [33] that allows users to anonymously perform the evaluation. From this feedback we improved the gene encoding strategy (changing it from an initial drop-down menu to a drag-and-drop interface), added a colour for each gene, and made the gene clearer (through the drag-and-drop interface). For the browser study, 18 participants were recruited via social media covering a range of backgrounds. Because of our anonymous system we do not know who completed our evaluation, but from conversations and where we advertised we conclude: eight were general public (not at the University) and four postgraduate and six undergraduate students (from a range of computing, nursing and psychology). Separately, the mobile study was conducted under controlled conditions using a single iPhone and Apple Watch, this study comprised two groups, the first was eight final year Computer Science students, the second group comprised nine members of the public.

Results. In the browser study the SUS survey was fully completed by 15 of the 18 participants. We calculate the mean SUS score to be 75.42 (Fig. 14) indicating a 'good' score according to Bangor et al. [3]. The minimum score was 40 while the maximum score was 92.5. The lowest score is interesting, because while they answered low values in the SUS each of their comments were extremely positive; perhaps they misunderstood how to complete the SUS Likert scale. For the mobile study all 17 participants completed the SUS survey. The mean score was 80.83 indicating a 'good' score. The minimum 32.5 while the maximum was 92.5. The low score for the mobile study was accompanied by largely positive comments except that the labelling of properties did not make much sense to the user. Along with the SUS, we can analyse the created visualisations.

The browser study visualisations. 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, 9.06 visualisations were created per participant within five minutes. The survey automatically time-stamps every action, and especially it automatically progresses to the SUS after five minutes. Therefore we are able to evaluate the duration of the design process. 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 (through self judgement) and the remaining 74 disliked.

The mobile Study visualisations. During the mobile study 19 par-

ticipants created a total of 156 visualisations with a minimum number of 3 and a maximum of 12, this averages to 8.21 visualisations per user. Within the given five minute period participants created, on average, one visualisation every 36.54 seconds. 106 of the visualisations were sent to the Apple Watch indicating that 67.9% of the visualisations created were liked. The binary choice between *like* and *dislike* forced participants to make clear decisions over the rendered visualisation and whether amendments of the gene were needed. 157 of the visualisations in the browser study and 111 during the mobile were created using a unique combination of gene parameters, this adds credence to our assertion that the model allows users to create many different designs in a small amount of time.

8 DISCUSSION AND CONCLUSIONS

We believe the path model has huge potential to create new visualisations, and by manipulating objects on the path by the envelope and filtering operations and storing the gene we are able to create a wide range of visual designs. In fact, participants in our evaluation said that the model was *"simple to use"*, another said *"very easy to use"*, and it *"could create some interesting designs very fast"*. Indeed, our evaluation demonstrated that participants were rapidly creating visualisations, between 4 and 15 in five minutes. Some participants in our evaluation said that they would like to use it for *"mobile or smart watch applications, maybe for health or fitness indications"* with another participant saying that it would *"integrate into smart devices where you could actively quickly design and change backgrounds for devices"*. Other participants said that it would be *"suited to a dashboard including other more detailed plots, or as a summary visualisation on a mobile application"*.

The concept of using the **visual flow** as the main underpinning structure for creating visualisations is a new way to contemplate visualisation design. While other researchers have considered paths or skeleton constructs (e.g., [6, 12]), they have done so to classify visualisations, or to analyse analysed the visual flow of pictures (e.g., [16, 59]), whereas Pereira et al. utilise a simple path [62]. However, our work places the *flowgraph* as the core visual skeleton to design visualisations. Like with the line-of-action lines of Reilly [50] the flowgraph provides an excellent and understandable structure to underpin visualisation designs. Integrating the flowgraph into visualisation design not only provides a structured framework but also enhances the creation of alternatives. By defining how elements flow and interact within a visualization, the flowgraph enables designers to create narratives, emphasize relationships, and guide viewer understanding intuitively. This approach has potential to support dynamic visual storytelling. Moreover, leveraging the flowgraph as a foundational design element encourages exploration and innovation in visual representation techniques, paving the way for new insights and discoveries.

One of the challenges for a designer, though, is that they need to think creatively, to first imagine what the path would be, and then decide on how to map the objects along the path. Most visualisations are designed from the data up; where the visualisation designs are created pragmatically through mapping data to change visual variables that are placed on the display. However, in our approach, developers need to conceptualise the visual flow, and use this to structure their visual design (though, much like a sketch artist would do). From our evaluation and case studies (especially) it is clear that some paths are easy to comprehend. It is clear that a bar chart has a linear flow, it is

clear that a pie chart would have a circular path; but for some designs the path will be less clear. What path would be best for a tree? What would be the best path to label points on the graph? Such complex paths can be approximated using multiple paths or several jumps in the path, but there may be other ways to achieve this. When planning or designing new visualisations the user needs to think about the design and iteratively refine their ideas, perhaps starting off by sketching designs [73] before implementing them. We believe a similar approach is needed here. Visual designs will emerge, and the developer needs to contemplate what the visual path will be. Sometimes the designer will create a path and realise that it was the wrong strategy. Developers will need to learn how to create the 'best' and most efficient path.

The gene provides a quick way to re-factor designs. We demonstrated this in our second case study. Once the path, envelope, object placement, filters and data mapping have been created, it is easy to swap out some of the parts to create a new design. In that case, turning a straight path into a circular one. Such a strategy would be difficult with traditionally built visualisations. The idea of the gene was positively discussed in our evaluation. Once participant said "*love how you can alter the final visualisation without having to start from scratch*". Some did say that some genes do create similar visualisations, but we believe this is the result of their use of the tool and choices they made. There was some confusion over some of our name choices, for example one participant wrote: "*..I did not know what an 'inline half' was before I created the visualisation and even though I liked the outcome*". When utilising the path editor, users have the freedom to define their preferred path.

One limitation with Genii is that it is not always easy to change from one visual flow to another. For instance, swapping between a straight path to create a circular visualisation using circular objects, into one that uses a circular path with straight objects currently requires the user to re-define the path and swap the mapped objects. But similar challenges exist when programming these visualisations from a library, and refactoring the code. We are currently improving our implementation to allow this pivoting operation to be simpler. There are other improvements we could make. One participant said "*I would have liked a textual or visual description of what each property would do to my visualisation.*" and another "*when there were many visualisations in the right hand panel, knowing which gene corresponded to which visualisation was tricky when I wanted to copy and modify them*". We are working to develop and improve the implementation.

There could be several extensions, or further ideas to our flowgraph model, that we could explore. **Paths-in-path** could simplify the creation of hierarchical structures. We could define several paths, name them, and use them later. This re-use could make path design easier. However, we did not include path-in-path because our flowpath model already allow jumps, can contain infinite vertices, the path can depend on complex data, which in turn can be used to provide the same functionality as paths-in-paths would provide. Another issue would be that adding such path-references and paths-in-paths would make the input parser more complex, and it is not necessarily requires. It is already possible to create an application to output more complex paths that could encode several complex jumps to that would be necessary to model a tree, treemap or other hierarchical visualisation. We also considered adding **logic into the path** itself. For instance, we already make decisions on the path which are based on the data. This is used to alternate the position of the objects on, or above the path, to change their colour and so forth. But it could be possible to embed logical decisions on the path. To achieve this functionality we would need to add ID labels to control jumps within the path itself. This would provide another way to build hierarchical (or other complex) structures. Applying logic to the path could allow element reuse, and recursion to occur within the design process. However, this would turn our path model from a scene description language into a programming language. From deterministic to non-deterministic. There has been a similar discussion within the scene graph community – where researchers have discussed the placement of logical statements within the scene [34, 93]. Much like standard scene graph implementations [29, 68, 88], our model does not include logic at the path level. But not having logical, variables or ID's

embedded with the path does not restrict complex paths to be created; we can still code a program to write more complex paths, which can then be used as before.

Another addition is to **add labels and legends**. One solution we contemplated was to add a specific path-type to layout this metadata. One path description for the data and another for the meta-information. However, because our path contains jumps, we can already achieve this functionality. Our solution is to append this information to the end of the path. The path jumps to each location to place text, in the same way as we describe a scatterplot visualisation, see Fig. 9. Alternatively labels could be added when the objects are placed – where the object becomes a multivariate object (containing geometry and text). The position of the text label could be determined by one of the numerous label placement strategies [1, 18]. Another way could be to couple the idea of logic within the path with label placement. Legends may also be placed along another path with decisions made at on construction or placement at any part of the model.

Our model allows the creation of many visualisations – some good, some bad – and by using the gene we are able to swap in/out parameters to discover or create new visualisations. Therefore, as one of the evaluation participants noticed, it is possible to create unsuitable visualisations. While not the focus of this paper, we do note that we are currently performing a more in depth evaluation of different visualisations designed by our method. With our builder interface, and even using the given (non- user-created) paths, we can combine and create thousands of parameters, and among them some will be poor visualisations. For instance, let's assume we want to apply a circle to a straight line path. By default, our circle segment has no data, and so it has a zero height; at the start of the process it will not necessarily look good. By adding default and random data we acknowledge that some early designs may not look good, and require effort on the part of the designer. But this is a typical solution to designing: "*the best way to a good idea is to have lots of ideas*" (Pauling, [20]). Subsequently, one solution could be to include design rules into our path model. Rules could make some automatic design decisions to the placement of objects such as used in graph-drawing or label placement algorithms [1] or perhaps delivered from machine learning, such as Draco [54].

In fact, this is one of the areas we want to expand. In future work we imagine automatically creating different genes, that will subsequently produce different visualisations. We aim to further explore the effectiveness of visualisations in a more in depth evaluation and use these evaluations to drive future gene generation to recommend good visualisations to the user. An important output from these future evaluations would also be to establish recommended envelope parameters for given paths. This concept of a gene allows us to think of visualisations as a small series of properties. With this we would like to explore the possibility of gene mutation and combination. Borrowing concepts from Genetic Algorithms, we would like to create an implementation which allows users to combine and refine visualisations creating new generations and allowing for the usage of mutations. Another future work idea is to develop into 3D and investigate different coordinate systems.

In this paper we have introduced a path-based design model, the Genii system (component library, builder and renderer). We also presented an in-depth description of the flowmodel. The model encourages people to think about the visualisation in terms of a visual path, define a flowpath, apply different envelopes, place objects and filters on the path to generate alternative visualisation designs. The visualisations are saved as SVG, which can used in other applications; and have demonstrated how the system can be applied for use on a smartwatch. We demonstrate the model and system works, by several case study applications. Where we re-create several visualisations in the literature and one from a sketch. We have performed a user evaluation, which supports that the model is understandable and can be used to create a range of visualisations. By encoding the path model into a gene we are able to swap in/out parameters to quickly create different visual depictions. Finally, we hope that our path model will be extended and developed by other researchers.

SUPPLEMENTAL MATERIALS

The Genii system (jamesjacko.github.io/genii/) is released under CC BY 4.0, and includes the component library doi:10.5281/zenodo.12571856, renderer doi:10.5281/zenodo.12571917 and builder doi:10.5281/zenodo.12571944.

FIGURE CREDITS

Credit all images to authors; Figs. 1 to 14, with the addition of accompanying thumbnails from Blascheck et al. [7] (top) and Brehmer et al. [13] (middle) in Fig. 12.

ACKNOWLEDGMENTS

REFERENCES

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label Placement by Maximum Independent set in Rectangles. *Comp. Geometry*, 11(3):209 – 218, 1998. doi: 10.1016/S0925-7721(98)00028-5

[2] R. Arnheim. *Art and visual perception: A psychology of the creative eye*. Univ of California Press, 1965.

[3] A. Bangor, P. Kortum, and J. Miller. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *J. Usability Studies*, 4(3):114–123, 2009.

[4] G. D. Battista et al. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice, 1st ed., 1998.

[5] F. Beck and D. Weiskopf. Word-Sized Graphics for Scientific Texts. *IEEE Trans. Visual Comput. Graphics*, 23(6):1576–1587, 2017. doi: 10.1109/TVCG.2017.2674958

[6] J. Bertin. *Graphics and Graphic Inf. Processing*. deGruyter, 1981.

[7] T. Blascheck et al. Glanceable Visualization: Studies of Data Comparison Performance on Smartwatches. *IEEE Trans. Visual Comput. Graphics*, 25(1):630–640, 2019. doi: 10.1109/TVCG.2018.2865142

[8] J. F. Blinn. A Generalization of Algebraic Surface Drawing. *Trans. Graph.*, 1(3):235–256, 1982. doi: 10.1145/357306.357310

[9] R. Borgo, J. Kehrer, et al. Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. In *Eurographics (STARs)*, pp. 39–63, 2013. doi: 10.2312/conf/EG2013/stars/039-063

[10] U. Brandes, B. Nick, B. Rockstroh, and A. Steffen. Gestaltlines. *CGF*, 32:171–180, 2013. doi: 10.1111/cgf.12104

[11] M. Brehmer, R. Kosara, and C. Hull. Generative Design Inspiration for Glyphs with Diatoms. *IEEE Trans. Visual Comput. Graphics*, 28(1):389–399, 2022. doi: 10.1109/TVCG.2021.3114792

[12] M. Brehmer, B. Lee, et al. Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE Trans. Visual Comput. Graphics*, 23(9):2151–2164, Sep. 2017. doi: 10.1109/TVCG.2016.2614803

[13] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. Visualizing Ranges over Time on Mobile Phones: A Task-Based Crowdsourced Evaluation. *IEEE Trans. Visual Comput. Graphics*, 25(1):619–629, Jan 2019. doi: 10.1109/TVCG.2018.2865234

[14] G. B. Bridgman. *Bridgman's Life Drawing*. Dover Publications, 1971.

[15] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[16] Z. Bylinskii, M. A. Borkin, et al. Eye Fixation Metrics for Large Scale Evaluation and Comparison of Information Visualizations. In *Proc.ETVIS*, pp. 235–255. Springer, 2017. doi: 10.1007/978-3-319-47024-5_14

[17] H. Chernoff. The Use of Faces to Represent Points in k-Dimensional Space Graphically. *JASA*, 68(342):361–368, 1973.

[18] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans.Graph.*, 14(3):203–232, 1995. doi: 10.1145/212332.212334

[19] M. Connor and P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Trans. Visual Comput. Graphics*, 16(4):599–608, 2010. doi: 10.1109/TVCG.2010.9

[20] F. Crick. The Impact of Linus Pauling on Molecular Biology. In *The Pauling Symposium: Special Collections, The Valley Library, Oregon State University, OR*, 1996.

[21] D. S. Ebert et al. Procedural Shape Generation for Multi-dimensional Data Visualization. *Comp.&Graph.*, 24(3):375 – 384, 2000. doi: 10.1016/S0097-8493(00)00033-9

[22] C. W. Eriksen and D. W. Schultz. Information processing in visual search: A continuous flow conception and experimental results. *Perc. & Psych.*, 25(4):249–263, 1979.

[23] J. Faragasso. *Mastering Drawing the Human Figure From Life, Memory, Imagination*. Dover Publications Inc, 2020.

[24] S. J. Fernstad and J. J. Westberg. To Explore What Isn't There—Glyph-Based Visualization for Analysis of Missing Values. *IEEE Trans. Visual Comput. Graphics*, 28(10):3513–3529, 2021. doi: 10.1109/TVCG.2021.3065124

[25] J. Fuchs, F. Fischer, et al. Evaluation of Alternative Glyph Designs for Time Series Data in a Small Multiple Setting. In *Proc. SIGCHI*, pp. 3237–3246. ACM, 2013. doi: 10.1145/2470654.2466443

[26] J. Fuchs, P. Isenberg, A. Bezerianos, and D. Keim. A Systematic Review of Experimental Studies on Data Glyphs. *IEEE Trans. Visual Comput. Graphics*, 23(7):1863–1879, 2017. doi: 10.1109/TVCG.2016.2549018

[27] T. Gerrits, C. Rössl, and H. Theisel. Towards Glyphs for Uncertain Symmetric Second-Order Tensors. *CGF*, 38(3):325–336, 2019. doi: 10.1111/cgf.13692

[28] P. Goffin, W. Willett, J. Fekete, and P. Isenberg. Exploring the Placement and Design of Word-Scale Visualizations. *IEEE Trans. Visual Comput. Graphics*, 20(12):2291–2300, 2014. doi: 10.1109/TVCG.2014.2346435

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE CVPR*, June 2016.

[30] R. Hu, B. Chen, et al. Shape-Driven Coordinate Ordering for Star Glyph Sets via Reinforcement Learning. *IEEE Trans. Visual Comput. Graphics*, 27(6):3034–3047, 2021. doi: 10.1109/TVCG.2021.3052167

[31] J. Jackson, P. Ritsos, and J. Roberts. Towards a Tool for the Creation of Micro-visualisations. In *Proc.CGVC*. EG, 2019. doi: 10.2312/cgvc.20191270

[32] J. Jackson, P. D. Ritsos, and J. C. Roberts. Creating Small Unit Based Glyph Visualisations. In *Posters IEEE VIS*, 2018.

[33] J. Jackson and J. C. Roberts. VisSurvey.js – A Web Based Javascript Application for Visualisation Evaluation User Studies. In *Posters IEEE VIS*, 2017.

[34] J. Johnson, R. Krishna, et al. Image Retrieval Using Scene Graphs. In *IEEE CCVPR*, 2015.

[35] D. Kammer, M. Keck, et al. Glyphboard: Visual Exploration of High-Dimensional Data Combining Glyphs with Dimensionality Reduction. *IEEE Trans. Visual Comput. Graphics*, 26(4):1661–1671, 2020. doi: 10.1109/TVCG.2020.2969060

[36] M. Keck and L. Engeln. Sparkle glyphs: A glyph design for the analysis of temporal multivariate audio features. In *Proc. AVI*, pp. 1–3, 2022. doi: 10.1145/3531073.3534491

[37] M. Khawatmi, Y. Steux, et al. ShapoGraphy: A User-Friendly Web Application for Creating Bespoke and Intuitive Visualisation of Biomedical Data. *Front. Bioinform.*, 2, July 2022. doi: 10.3389/fbinf.2022.788607

[38] G. Kindlmann and C. Westin. Diffusion Tensor Visualization with Glyph Packing. *IEEE Trans. Visual Comput. Graphics*, 12(5):1329–1336, 2006. doi: 10.1109/TVCG.2006.134

[39] K. Koc, A. S. McGough, and S. Johansson Fernstad. PeaGlyph: Glyph design for investigation of balanced data structures. *Inf. Visualization*, 21(1):74–92, 2022. doi: 10.1177/14738716211050602

[40] P. A. Legg et al. MatchPad: Interactive Glyph-Based Visualization for Real-Time Sports Performance Analysis. *CGF*, 31(3):1255–1264, 2012. doi: 10.1111/j.1467-8659.2012.03118.x

[41] P. A. Legg, E. Maguire, S. Walton, and M. Chen. Glyph Visualization: A Fail-Safe Design Scheme Based on Quasi-Hamming Distances. *CG&A*, 37(2):31–41, 2017. doi: 10.1109/MCG.2016.66

[42] G. Li and X. Yuan. GoTreeScape: Navigate and Explore the Tree Visualization Design Space. *IEEE Trans. Visual Comput. Graphics*, 29(12):5451–5467, 2023. doi: 10.1109/TVCG.2022.3215070

[43] S. Li, R. J. Crouser, G. Griffin, C. Gramazio, H.-J. Schulz, H. Childs, and R. Chang. Exploring hierarchical visualization designs using phylogenetic trees. In *Proc. VDA*, vol. 9397, pp. 68–81. SPIE, 2015.

[44] Y.-T. Liu, Y.-C. Guo, et al. Learning implicit glyph shape representation. *IEEE Trans. Visual Comput. Graphics*, pp. 1–12, 2022. doi: 10.1109/TVCG.2022.3183400

[45] Z. Liu, J. Thompson, et al. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proc.CHI*. ACM, 2018. doi: 10.1145/3173574.3173697

[46] A. Loomis. *Figure Drawing for All It's Worth (How to draw and paint)*. Viking Adult, 1943.

[47] D. Lopes, J. Correia, and P. Machado. Adea – evolving glyphs for aiding creativity in typeface design. In *Proc. GE Comp.*, pp. 97–98, 2020. doi: doi.org/10.1145/3377929.3389964

[48] E. Maguire et al. Taxonomy-Based Glyph Design. *IEEE Trans. Visual Comput. Graphics*, 18(12):2603–2612, 2012. doi: 10.1109/TVCG.2012.271

[49] E. Maguire et al. Visual Compression of Workflow Visualizations with Automated Detection of Macro Motifs. *IEEE Trans. Visual Comput. Graphics*, 19(12):2576–2585, 2013. doi: 10.1109/TVCG.2013.225

[50] M. D. Mattesi. *Force: Dynamic Life Drawing for Animators*. CRC Press, 3rd Ed., 2017.

[51] L. E. Matzen et al. Patterns of Attention: How data Visualizations are Read. In *Proc. Augmented Cognition*, pp. 176–191. Springer, 2017. doi: 10.1007/978-3-319-58628-1_15

[52] L. E. Matzen et al. Data Visualization Saliency Model: A Tool for Evaluating Abstract Data Visualizations. *IEEE Trans. Visual Comput. Graphics*, 24(1):563–573, 2018. doi: 10.1109/TVCG.2017.2743939

[53] L. McNabb and R. S. Laramee. Multivariate Maps—A Glyph-Placement Algorithm to Support Multivariate Geospatial Visualization. *Information*, 10(10), 2019. doi: 10.3390/info10100302

[54] D. Moritz et al. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Trans. Visual Comput. Graphics*, 25(1):438–448, 2019. doi: 10.1109/TVCG.2018.2865240

[55] M. Naveh-Benjamin and R. G. Pachella. The Effect of Complexity on Interpreting "Chernoff" Faces. *Human factors*, 24(1):11–18, 1982.

[56] J. Nielsen and K. Pernice. *Eyetracking Web Usability*. New Riders Publishing, Thousand Oaks, CA, USA, 1st ed., 2009.

[57] A. Nuñez-Alcover et al. Glyph and position classification of music symbols in early music manuscripts. In *IbPRIA, Madrid, Spain*, pp. 159–168. Springer, 2019. doi: 10.1007/978-3-030-31321-0_14

[58] A. Offenwanger, M. Brehmer, et al. TimeSplines: Sketch-Based Authoring of Flexible and Idiosyncratic Timelines. *IEEE Trans. Visual Comput. Graphics*, 30(1):34–44, 2024. doi: 10.1109/TVCG.2023.3326520

[59] X. Pang, Y. Cao, R. W. Lau, and A. B. Chan. Directing User Attention Via Visual Flow on Web Designs. *ACM Trans.Graph.*, 35(6):240, 2016. doi: 10.1145/2980179.298242

[60] M. Patel and D. H. Laidlaw. Visualization of 3D Stress Tensor Fields using Superquadric Glyphs on Displacement Streamlines. *IEEE Trans. Visual Comput. Graphics*, 27(7):3264–3276, 2021. doi: 10.1109/TVCG.2020.2968911

[61] J. Pearlman, P. Rheingans, and M. des Jardins. Visualizing Diversity and Depth over a Set of Objects. *CG&A*, 27(5):35–45, Sep. 2007. doi: 10.1109/MCG.2007.139

[62] F. A. Pereira, T. Martins, et al. Generative Type Design: Creating Glyphs from Typographical Skeletons. In *Proc. Dig. Int. Arts*, pp. 1–8, October 2019. doi: 10.1145/3359852.3359866

[63] C. Perin, R. Vuillemot, and J. Fekete. SoccerStories: A Kick-off for Visual Soccer Analysis. *IEEE Trans. Visual Comput. Graphics*, 19(12):2506–2515, Dec 2013. doi: 10.1109/TVCG.2013.192

[64] A. H. I. Pires et al. A Summarization Glyph for Sets of Unreadable Visual Items in Treemaps. In *Proc IV*, pp. 242–247. IEEE, 2020. doi: 10.1109/IV51561.2020.00047

[65] D. Presnov and A. Kolb. Glyph from Icon – Automated Generation of Metaphoric Glyphs. *CoRR*, 2022. doi: 10.48550/arXiv.2206.05061

[66] D. Rees, R. S. Laramee, et al. Agentvis: Visual analysis of agent behavior with hierarchical glyphs. *IEEE Trans. Visual Comput. Graphics*, 27(9):3626–3643, 2021. doi: 10.1109/TVCG.2020.2985923

[67] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive Interactive Design of Information Visualizations. *IEEE Trans. Visual Comput. Graphics*, 20(12):2092–2101, Dec 2014. doi: 10.1109/TVCG.2014.2346291

[68] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In C. Cortes et al., eds., *Proc. Neural Inf. Proc. Systems*, pp. 91–99, 2015.

[69] J. C. Roberts. Visualization Display Models – Ways to Classify Visual Representations. *Comp. Int. Design & Const.*, 2(4):241—250, 2000.

[70] J. C. Roberts. State of the art: Coordinated multiple views in exploratory visualization. In *Proc.CMV*, pp. 61–71, 2007. doi: 10.1109/CMV.2007.20

[71] J. C. Roberts, P. Butcher, A. Sherlock, and S. Nason. Explanatory journeys: Visualising to understand and explain administrative justice paths of redress. *IEEE Trans. Visual Comput. Graphics*, 28(1):518–528, 2022. doi: 10.1109/TVCG.2021.3114818

[72] J. C. Roberts and K. Franklin. Haptic glyphs (hlyphs) - structured haptic objects for haptic visualization. In *Proc. World Haptics*, pp. 369–374, 2005. doi: 10.1109/WHC.2005.68

[73] J. C. Roberts, C. Headleand, and P. D. Ritsos. Sketching Designs Using the Five Design-Sheet Methodology. *IEEE Trans. Visual Comput. Graphics*, 22:419–428, 2016. doi: 10.1109/TVCG.2015.2467271

[74] T. Ropinski, S. Oeltze, and B. Preim. Survey of Glyph-based Visualization Techniques for Spatial Multivariate Medical Data. *Comp.&Graph.*, 35(2):392 – 401, 2011. doi: 10.1016/j.cag.2011.01.011

[75] T. Saito, H. N. Miyamura, et al. Two-tone Pseudo Coloring: Compact Visualization for One-dimensional Data. In *Proc. InfoVis*, pp. 173–180, 2005. doi: 10.1109/INFVIS.2005.1532144

[76] A. Satyanarayan and J. Heer. Lyra: An Interactive Visualization Design Environment. *CGF*, 33(3):351–360, 2014. doi: 10.1111/cgf.12391

[77] W. Schroeder, K. Martin, and W. Lorensen. The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization. In *Proc. Vis*, pp. 93–100, 1996. doi: 10.1109/VISUAL.1996.567752

[78] H.-J. Schulz and S. Hadlak. Preset-based Generation and Exploration of Visualization Designs. *J. Vis. Lang. & Comp.*, 31:9–29, 2015. doi: 10.1016/j.jvlc.2015.09.004

[79] C. D. Schulze, N. Wechselberg, and R. von Hanxleden. Edge label placement in layered graph drawing. In *Diag. Repr. & Inf.*, pp. 71–78. Springer, Cham, 2018. doi: 10.1007/978-3-319-91376-6_10

[80] Y. Sekse Kristiansen and S. Bruckner. Visception: An Interactive Visual Framework for Nested Visualization Design. *Comp.&Graph.*, 92:13–27, 2020. doi: 10.1016/j.cag.2020.08.007

[81] A. Slingsby, J. Dykes, and J. Wood. Configuring Hierarchical Layouts to Address Research Questions. *IEEE Trans. Visual Comput. Graphics*, 15(6):977–984, 2009. doi: 10.1109/TVCG.2009.128

[82] J. Suschnigg, B. Mutlu, et al. Visual Exploration of Anomalies in Cyclic Time Series Data with Matrix and Glyph Representations. *Big Data Research*, 26, 2021. doi: 10.1016/j.bdr.2021.100251

[83] T. Tsandilas. StructGraphics: Flexible Visualization Design through Data-Agnostic and Reusable Graphical Structures. *IEEE Trans. Visual Comput. Graphics*, 27(2):315–325, 2021. doi: 10.1109/TVCG.2020.3030476

[84] E. R. Tufte. *Visual Explanations*. Graphics Press, Connecticut, 1997.

[85] E. R. Tufte. *The Visual Display of Quantitative Information*, vol. 2. Connecticut, 2001.

[86] C. van Onzenoodt, P.-P. Vázquez, and T. Ropinski. Out of the Plane: Flower Vs. Star Glyphs to Support High-Dimensional Exploration in Two-Dimensional Embeddings. *IEEE Trans. Visual Comput. Graphics*, pp. 1–15, 2022. doi: 10.1109/TVCG.2022.3216919

[87] R. Vuillemot and J. Boy. Structuring Visualization Mock-Ups at the Graphical Level by Dividing the Display Space. *IEEE Trans. Visual Comput. Graphics*, 24(1):424–434, 2018. doi: 10.1109/TVCG.2017.2743998

[88] G. Wang, S. Li, S. Wang, B. Lu, and W. Li. ViWoSG: A Distributed Scene Graph of Ultramassive Distributed Virtual Environments. *Sci.in China Series*, 52:457–469, 2009. doi: 10.1007/s11432-009-0071-3

[89] M. O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Info. Vis.*, pp. 194–210, 2002. doi: 10.1057/palgrave.ivs.9500025

[90] C. Ware. *Information Visualization:.*. Kaufmann, 2nd ed., 2004.

[91] W. Willett, J. Heer, and M. Agrawala. Scented Widgets: Improving Navigation Cues with Embedded Visualizations. *IEEE Trans. Visual Comput. Graphics*, 13(6):1129–1136, 2007. doi: 10.1109/TVCG.2007.70589

[92] S. won Park. Generating Novel Glyph Without Human Data by Learning to Communicate. In *Proc. MLfC*. Springer, 2020. arxiv.org/pdf/2010.04402.

[93] D. Xu et al. Scene Graph Generation by Iterative Message Passing. In *IEEE CVPR*, July 2017. doi: 10.1109/CVPR.2017.330

[94] M. Yamamoto, G. Camara, and L. A. N. Lorena. Tabu Search Heuristic for Point-Feature Cartographic Label Placement. *GeoInformatica*, 6(1):77–90, 2002. doi: 10.1023/A:1013720231747

[95] L. Ying, X. Shu, et al. MetaGlyph: Automatic Generation of Metaphoric Glyph-based Visualization. *IEEE Trans. Visual Comput. Graphics*, 29(1):331–341, 2023. doi: 10.1109/TVCG.2022.3209447

[96] L. Ying, T. Tangl, et al. GlyphCreator: Towards Example-based Automatic Generation of Circular Glyphs. *IEEE Trans. Visual Comput. Graphics*, 28(1):400–410, 2022. doi: 10.1109/TVCG.2021.3114877

[97] S. Zeller, F. Samsel, and L. Bartram. Affective, Hand-Sculpted Glyph Forms for Engaging and Expressive Scientific Visualization. In *VISAP*, pp. 127–136. IEEE, 2022. doi: 10.1109/VISAP57411.2022.00025

[98] C. Zhu-Tian, Y. Su, Y. Wang, et al. MARVisT: Authoring Glyph-Based Visualization in Mobile Augmented Reality. *IEEE Trans. Visual Comput. Graphics*, 26(8):2645–2658, 2020. doi: 10.1109/TVCG.2019.2892415