Glyphaestus: Interactive Design of Parametric Glyphs with Hierarchical Composition

Vincent Bonczak
vincent.bonczak@inria.fr
Université Paris-Saclay, CNRS, Inria
Gif-sur-Yvette, France

Michel Beaudouin-Lafon
mbl@lisn.fr
Université Paris-Saclay, CNRS, Inria
Gif-sur-Yvette, France

Theophanis Tsandilas
theophanis.tsandilas@inria.fr
Université Paris-Saclay, CNRS, Inria
Gif-sur-Yvette, France

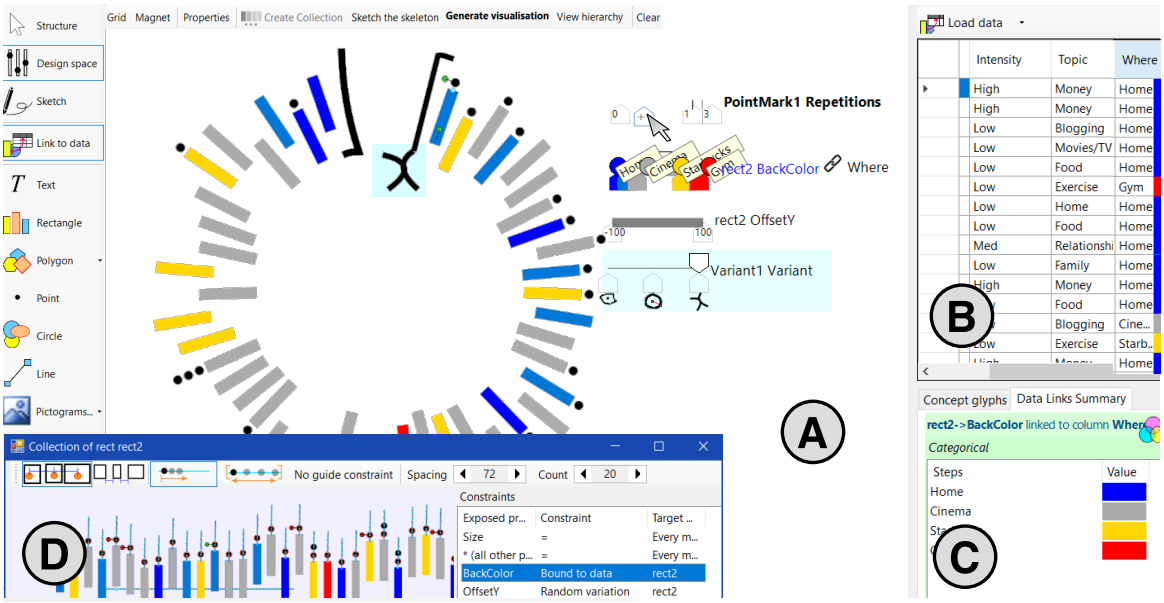


Figure 1: The Glyphaestus main interface reproducing G. Lupi's Week 8 visualization design [16]. In the canvas (A), users construct the glyph structure, expose visual properties as sliders, and define variation ranges and interpolations. The data grid (B) supports rapid data binding via drag-and-drop operations and contextual menus. Bindings can be modified from a summary view (C) or directly on the canvas through draggable plugs on the sliders. Collections nested within a glyph can be configured using the collection view (D), which presents a linear representation for adjusting layout and property constraints.

Abstract

The expressivity of hand-drawn and design-oriented visualizations stems from the rich composition, hierarchical organization, and controlled variation of glyphs. However, existing visualization authoring tools offer limited support for constructing such glyphs: they treat glyphs as flat assemblies of marks, support limited parametrization, and rely on procedural workflows that hinder exploration.

We present Glyphaestus, an interactive system for designing parametric, drawing-based glyphs with hierarchical composition and structured variability. Glyphs are compositions in which designers define reusable components, expose visual parameters, and specify constraints across multiple levels. This enables designers to move from graphical construction to data semantics prior to data binding. We report on a preliminary user study exploring how Glyphaestus expands the expressive design space of glyph-based visualization and supports the transition from graphical representations to data-driven concepts.

Keywords

Visualization authoring systems, expressive data visualization, parametric design, glyph-design tools, graphical constraints

ACM Reference Format:
Vincent Bonczak, Michel Beaudouin-Lafon, and Theophanis Tsandilas. 2018. Glyphaestus: Interactive Design of Parametric Glyphs with Hierarchical Composition. In *Proceedings of (Conference acronym 'XX)*. ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Drawing enables designers to create visual representations that are not only informative but also deeply personal, expressive, and conceptually rich. In personal data visualization practices such as Dear Data by Giorgia Lupi and Stefanie Posavec [16], glyphs are

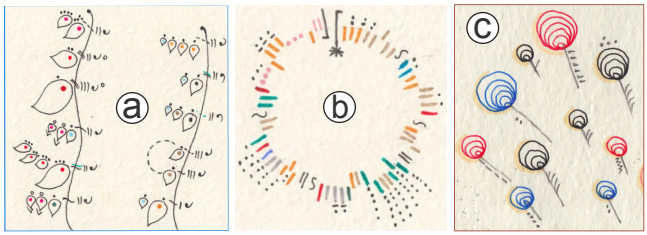


Figure 2: Examples from Dear Data [16] with complex glyph designs: *Week 5* (a), *Week 8* (b), and *Week 36* (c).

not merely visual encodings—their shapes, structures, and internal organization embody how designers interpret and frame data as meaningful constructs. These examples demonstrate that expressiveness arises not only from stylistic freedom, but also from composition and parametrization of graphical forms.

Existing drawing-based visualization systems such as DataInk [42], Data-Driven Guides [12], and DataGarden [22] have made important advances toward supporting expressive visualization authoring. Likewise, powerful user interfaces and grammars, such as DataIllustrator [15], Charticulator [26], StructGraphics [36], MSC [13], and PiCCL [34], provide sophisticated mechanisms for composing graphical structures. However, these tools rarely frame glyph design itself as a conceptual modelling activity—one in which designers explicitly think of glyphs as structured data concepts with reusable internal hierarchies and explicit parametrization. Moreover, many of these systems rely on highly procedural workflows [29], which can limit design exploration and make it difficult to restructure existing designs.

In this work, we introduce Glyphaestus[1], an interactive system for parametric, drawing-based glyph design that treats glyph construction as a conceptual modelling process. Our approach encourages designers to think of glyphs not just as visual encodings, but as structured entities capable of containing collections, nested compositions, and multiple hierarchical levels. Our system supports composition operators similar in spirit to those found in MSC [13] and PiCCL [34], while enabling a flexible interaction-based workflow that combines top-down and bottom-up design moves. Designers can iteratively refine the composition of a glyph, and these changes propagate consistently across associated collections. This bidirectional workflow allows for both structural rethinking and incremental refinement, supporting exploratory and expressive design processes.

Our key contributions include the unification of composition and layout through interactive constraint-specification objects, such as structural guides that define the layout of nested collections, and the integration of design principles inspired by programming, such as typing and encapsulation. By combining hierarchical composition, explicit parametrization, and freeform graphical primitives within a unified interaction framework, Glyphaestus expands the expressive design space of parametric glyph-based visualizations. We also present the results of a qualitative study with 14 participants using Glyphaestus, and conclude with a discussion of the system's capabilities and limitations through representative examples.

[1]The name refers to Hephaestus, the Greek god of artisans and craftsmen, who forged the shield of Achilles, renowned for its spectacular imagery.

2 Related work

Our system design is informed by prior work on visualization authoring and interactive tools for defining graphical constraints.

2.1 Visualization authoring systems

A wide range of systems support the creation of data visualizations, each offering different levels of expressivity. Chart-building tools with integrated spreadsheets, such as Excel, PowerBI, and Tableau, enable the rapid generation of visualizations from data with minimal effort. However, they offer limited customization beyond basic visual properties such as fonts and colour. As observed in prior work [2], visualization authors often switch between multiple tools to reach a desired result, frequently relying on graphical design tools such as Adobe Illustrator.

An active line of research has introduced drawing capabilities into visualization systems, each focusing on different aspects of expressivity. Lyra [28], Data Illustrator [15], and Charticulator [26] support visualization authoring with nested layouts. InfoNice [38] enables the creation of infographics with custom marks, while Data-Driven Guides [12] support the use of organic, deformable shapes. StructGraphics [36] allows designers to define parametric graphical spaces and constraints without requiring a predefined dataset.

Other work draws inspiration from hand-drawn visualization practices such as Dear Data [16, 33], aiming to support more personal and idiosyncratic visual vocabularies. Systems such as DataInk [42], DataSelfie [11], TimeSplines [21], and DataGarden [22] exemplify this approach. By relying on sketch-based input, they enable the rapid creation of freeform visuals, closer to pen-and-paper drawing. CompSketch [35] further supports freeform variations by allowing users to specify data bindings through annotations, fostering the exploration of alternative mappings. However, this expressive freedom comes at a cost: it becomes more difficult to formally represent underlying structures and generalize them beyond what has been explicitly drawn. Finally, other systems [25] automatically infer data from hand-drawn glyphs based on user-defined parametric templates.

While each system targets specific use cases, none explicitly focuses on glyph design and none fully supports the complex, hierarchical glyph structures observed in many Dear Data examples, e.g., those in Figure 2. Diatoms [4] supports generative design for data glyphs, but treats glyphs as flat groups of marks without internal structure. In contrast, our work focuses on supporting the iterative design of glyphs with hierarchical composition.

2.2 Toolkits and visualization grammars

Low-level graphics libraries provide basic drawing primitives but require substantial effort to build data visualizations. Libraries such as D3 [3] facilitate data binding but still rely on programming expertise. Higher-level approaches introduce domain-specific or declarative languages for visualization, including Diagrams [44], Penrose [43], Bluefish [23], ggplot2 [39], Vega [31], and Vega-Lite [30]. More recent systems, such as Atlas [14], MSC [13], PiCCL [34], and GoFish [24], extend these ideas with compositional operators and constraint-based mechanisms for constructing visualizations.

While widely used, these approaches primarily target analytical visualization and prioritize readability, limiting their expressiveness

for creative or personal visualizations. Our work builds on their concepts while extending their expressive capabilities.

2.3 Manipulating graphical constraints

While formal languages can describe visual representations, a central challenge in HCI research is to *reify* [1] their internal structure and constraints as interactive objects.

Most visualization authoring tool, e.g., Data Illustrator [15], Charticulator [26], and DataGarden [22], rely on hierarchical organizations of visual elements, but differ in how they support the specification of graphical constraints. For example, Charticulator defines equality constraints between properties, resolved through a constraint solver, while Data Illustrator relies on *lazy data binding* to link graphical properties. StructGraphics [36], in contrast, introduces a hierarchical *property-sharing* mechanism that operates independently of any predefined data schema. Visualization systems also provide tools for automatic layout of visual elements, such as *scaffolds* [4, 26] or *spines* [22]. Data-Driven Guides [12] introduce *data guides*, which not only encode data but also control the geometry of non-standard shapes. We aim to unify these concepts to support greater expressivity with fewer interaction techniques.

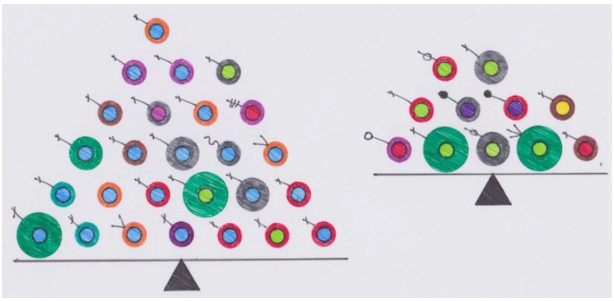
Beyond visualization, HCI research has proposed a variety of approaches for manipulating graphical properties. Hoarau and Conversy [8] explore interactive tools for sharing properties across objects. Object-Oriented Drawing [40] introduces *attribute objects* that expose and manipulate properties directly, while Collection Objects [41] extend this approach to groups of elements.

Other work focuses on expressing constraints between properties. StickyLines [6] represents alignment and distribution as persistent, manipulable properties, and introduces *tweaking* to support local variations. Graphical Substrates [18] enable designers to define layout constraints through visual programming, while Para [9] provides mechanisms for expressing functional constraints in procedural design. DrawingTransforms [7] further allow users to define procedural transformations through drawing. Finally, Mackay and Beaudouin-Lafon [17] unify many of these approaches through the notion of *interaction substrates*, which reify objects and their constraints into interactive object constructs. Our approach is strongly inspired by these principles.

3 Conceptual gaps in existing systems

To ground our discussion, we refer to the visualization of negative thoughts by Andy Kriebel [33] in Figure 3. The design comprises two classes of graphical objects at two hierarchical levels: (i) individual instances of negative thoughts, represented as decorated concentric circles  , and (ii) collections of thoughts, represented by a balance icon  .

A desirable quality of a design-oriented authoring tool is to let designers conceptualize and parametrize these glyph representations independently of one another, without requiring immediate binding to concrete data or predefined mappings. Designers should also be able to combine them flexibly—for example, by embedding individual negative-thought glyphs within the collection structure—while retaining the ability to iteratively refine each component. This includes introducing new parameters, such as encoding the location of a negative thought (e.g., at home or in a coffee shop)



	Location	With whom	Topic	Stress level
	Home	Beth	food	low
	Starbucks	Beth	money	high
	Bentall's	Oscar	relations	low

Figure 3: Visualization of inward (left) and outward (right) negative thoughts by Andy Kriebel [33] (Week 38). We show the data encodings used in the inner glyphs.

through custom mark variants  , without disrupting the overall composition.

Although several of the systems mentioned above adopt a graphics-driven approach and often explicitly target design-driven workflows, they do not adequately support such flexibility. We identify three limitations.

Glyph composition. In many systems [13, 15, 26, 28, 42], a glyph is defined as a flat grouping of marks at a single hierarchical level. StructGraphics [36] allows glyph nesting but does not support collections nested within glyphs. Conversely, Data-Driven Guides [12] enables multi-level structures, but these must be constructed manually by duplicating and attaching guides. The system lacks an internal representation of hierarchy and does not generalize well to new data. DataGarden [22] introduces hierarchical glyphs whose sub-glyphs can themselves become parents of collections, enabling nested multi-level visualizations. However, it assumes that all children of a node are of the same type and thus, for example, fails to express heterogeneous sub-collections, such as those in Figure 2.

PiCCL [34] and, to some extent, GoFish [24] support collections nested under glyphs (or marks), but they are grammar-based systems without direct-manipulation design tools.

Graphical constraints. The glyphs in Figure 3 are structured through constraints that govern both spatial relationships and other visual properties such as size and colour. Charticulator [26] and MSC [13] provide limited constraint mechanisms, primarily for relative positioning. PiCCL [34] supports richer geometric constraints, such as length and angle, while StructGraphics [36] introduces a more general *property-sharing* mechanism linking properties across elements. However, these systems mainly support equality-based constraints and lack mechanisms for expressing more complex relationships, such as functional dependencies between properties (e.g., the proportional relationships among circle sizes  in Figure 2c). More expressive constraint systems exist in design tools such as Linkify [18], Para [9], and Apparatus [32], but integrating such capabilities into data-driven workflows without requiring programming remains a challenge. Additionally, while most systems focus on layouts that position glyphs globally, we emphasize constraints governing the internal layout of glyph components.

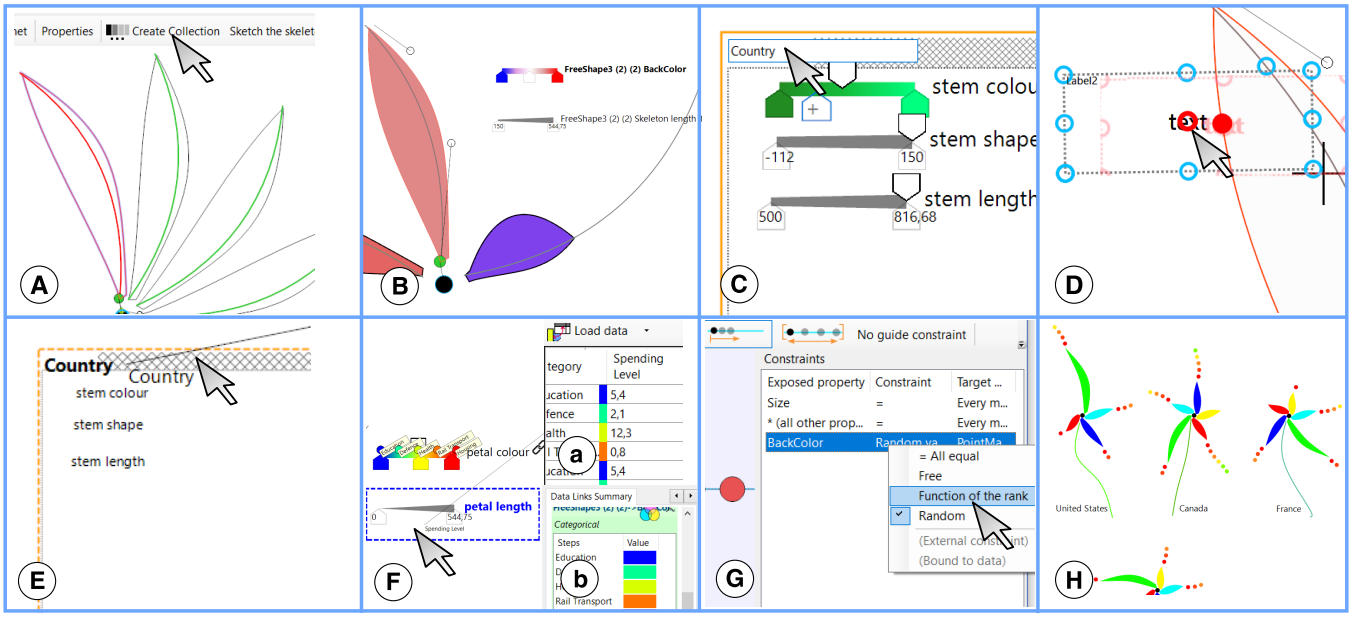


Figure 4: The visualization creation workflow for Denise. (A) She sketches petals and arranges them in a radial collection. (B) Background colour and length of petals are exposed as sliders: variations of these two properties across the collection appear. (C) Denise encapsulates the whole glyph and names the concept it represents. (D) She adds a label to one of the petals to refine the design of the collection. (E) She matches the Country data column with the concept she created in the canvas. (F) After having renamed the properties, she binds the category to the petal colour and the *Spending level* to the length of the petal. (G) Finally, she chooses a function-of-the-rank colour constraint for her collection of points. (H) The final render shows all the glyphs (one per country), with random variations for the stem colour and shape.

Creation workflow. The glyphs in Figure 3 represent data concepts (negative thoughts). In systems such as Lyra, Data Illustrator, and Charticulator, these concepts are introduced through bindings between visual channels and data attributes [27]. This approach requires a predefined data structure and tightly couples design to data, limiting flexibility and making iterative changes difficult. Supporting design exploration requires more flexible workflows, including a bottom-up construction [19], as in iVoLVER [20], partial or deferred data mappings, as in StructGraphics [37] and DataGarden [22], and generative approaches to glyph design, as in Diatoms [4]. Our goal is to build on these ideas while enabling greater flexibility in structuring and refining glyphs.

4 Designing data glyphs with Glyphaestus

Glyphaestus addresses the above challenges through three key capabilities:

(1) Interactive composition and parametrization of hierarchical glyphs, including nested collections and their associated constraints;
(2) Flexible design workflows that support multiple construction paths and iterative refinement; and
(3) Progressive conceptual modelling of glyphs in interaction with data.

As shown in Figure 1, the user interface comprises a glyph construction canvas (A), where designers can draw shapes, assemble them into hierarchical glyph structures, and define visual dimensions of interest. Glyphaestus also provides panels (B, C) for exploring and visualizing mappings between data attributes and these dimensions. Central to the system is a collection viewer (D), which enables users to configure the layout of sub-glyphs within embedded collections and specify property constraints. In addition, Glyphaestus supports generative design by allowing users to explore variations of a glyph, either driven by data or through expressive randomization.

Illustrative scenario. To illustrate how Glyphaestus supports glyph design, we present a usage scenario (accompanied by a video in the supplementary materials). The scenario follows Denise, a data journalist preparing an article on public funding across G7 countries. Denise wants to design an original visualization that enables readers to compare the relative share of GDP allocated to different sectors, such as transportation, environment, and education. Her dataset is structured around *Country*, *Category of Spending*, *Spending Level (2025)*, and *% Spending Change (since 2015)*.

Rather than using a conventional bar chart, Denise seeks a more expressive representation inspired by the OECD Better Life Index chart[2]. She decides to represent each country as a flower glyph, where petals encode different spending categories through variations in colour and size. At this stage, she focuses on designing the glyph itself—defining its structure and potential variations—before binding it to data. She also wants to introduce controlled variability across instances to avoid repetitive visual patterns. We now present the main steps of her creation workflow.

[2]See www.oecd.org/en/data/tools/well-being-data-monitor/better-life-index.html

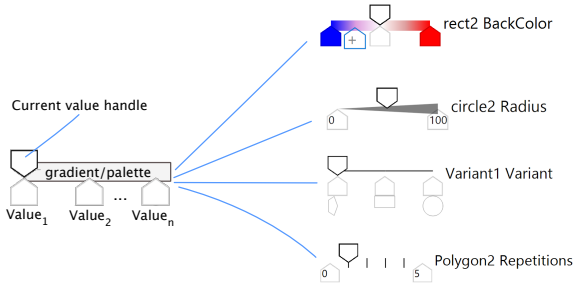


Figure 5: The generic slider concept and a selection of four property types: colour, continuous quantity, object and count.

4.1 Adding shapes and variables

Denise starts by sketching several candidate petal shapes and explores a radial collection layout to organize them (Figure 4A). This allows her to compare alternatives and select a preferred design while keeping others available for reuse. The radial layout was created because the root is a point mark: in the interface, the collection layout is selected based on the dimension (1D versus 2D) and the type of the root (see Figure 6).

She then begins to parameterize the glyph by *exposing* visual properties as interactive variables. She switches to the *Design Space* mode, right-clicks on the glyph, and exposes properties such as *skeleton length* and *background colour* (Figure 4B). Skeletons relate to the notion of *data-driven guides* [12], allowing designers to control the size of freeform shapes. However, unlike straight guides, skeletons in Glyphaestus can take arbitrary forms, defining a range of variation along their path—for example, as the curved skeleton of the purple petal in Figure 4B. As we discuss in Section 5, this mechanism generalizes to collections, where arbitrary paths can also serve as layout constraints.

Once exposed, properties become manipulable parameters, represented as interactive sliders directly on the canvas (Figure 4C). All parameters share a unified representation (Figure 5), with inputs and visual guides adapted to the type of visual variable. Because these parameters are defined on a shape already replicated within a collection, changes immediately propagate across all instances. This enables Denise to explore variations of the glyph in context, supporting both local refinement and global design exploration. She then draws the stem and, to reduce visual repetition, exposes its length and colour as parameters, introducing controlled random variation across shades of green. She decides to switch back to her previous petal design, using the menu to replace the collection template. The collection retains all its geometric constraints while updating to the new mark.

4.2 Adding semantics through concepts

Denise next assigns semantics to the glyph by defining it as a *concept* of interest (Figure 4C), namely a "Country". This step makes the glyph self-descriptive and facilitates reuse. She then realizes that the petals lack labels. She adds a label and anchors it to a single petal within the collection (Figure 4D). Once anchored, the label is replicated across all instances, reflecting the shared structure of the collection. In Glyphaestus, elements within a collection share both a common structure and a common design space (i.e., parameter variations). We refer to them as belonging to the same *type*.

Finally, she adds a label for the country itself. This label is instantiated once per "Country" glyph and anchored to the collection guide. To evaluate her design, Denise inspects the result in the visualization view and further iterates to explore visual variations. When no data binding is defined for a given exposed property, the system chooses a random value from its slider range, different for each glyph instance.

4.3 Data binding

Denise loads the dataset from a file into the data panel. To define the grouping of rows, she drags the *Country* column and drops it onto the previously created "Country" concept (highlighted with an orange dashed border in Figure 4E). This action establishes a mapping between unique values in the dataset and instances of the concept-glyph: each "Country" glyph is now mapped to a unique country value.

Finally, she links the *Spending Level (2025)* to the length of the petal by dragging the corresponding column header onto the parameter slider (indicated by a blue dashed border, Figure 4F). For clarity, she renames the associated parameters to better reflect their semantic meaning.

4.4 Iterating on the design

Denise considers encoding the *% Spending Change* using the length of a curve anchored at the tip of each petal. However, after sketching an initial curve, she finds the encoding difficult to interpret, as the curve originates from different positions on each petal, making comparisons challenging. To improve readability, she instead creates a collection of points along the curve, using the curve itself as the skeleton of the collection. She exposes the *Repetition* parameter and maps it to the percentage change. She also exposes the points' background colour, enabling her to experiment with different constraints in the collection viewer (Figure 4G).

She ultimately selects a *function-of-rank* constraint, allowing point colours to vary progressively from red to green (configured directly through the slider gradient) as the number of repetitions increases. Other types of constraints include *random* to introduce variations, as used previously for the stem colour, *all equal* to link all replicated glyphs for this property to the same value, and *free* (no constraint, so the property values are set at design-time).

Denise finds that this encoding more effectively conveys decreases and increases in spending. She then generates the final rendering using the visualization view (Figure 4H).

5 Glyphaestus's conceptual framework

This section details the core concepts of Glyphaestus, including glyph composition, visual property parametrization, and constraints.

5.1 Composition grammar

As in other visualization grammars [13, 34], we define glyphs as compositions of one or more marks. A formal description of Glyphaestus's grammar is presented below, where we highlight first-class objects in **blue**:

```
VisualElement :- Mark | Glyph | Collection
        Glyph :- VisualElement |
                 anchor(parent: VisualElement,
```

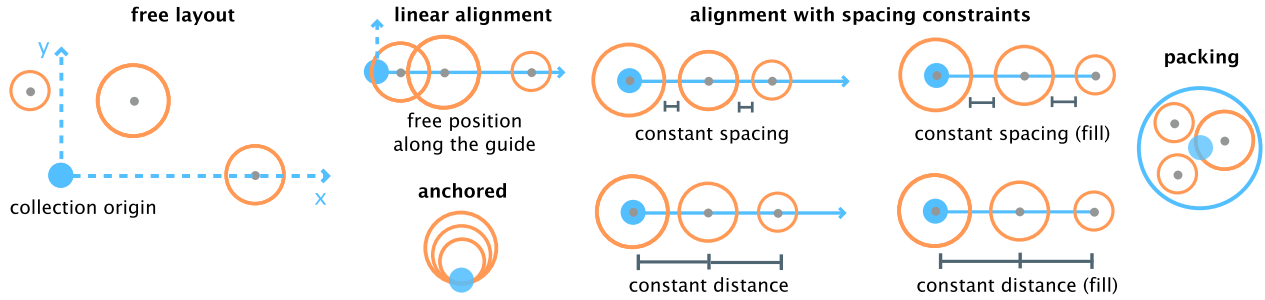


Figure 6: Strategies for arranging a collection of glyphs, each applying different constraints on their positioning (x, y) relative to the coordinate system defined by the collection.

```
                child: VisualElement, PropConstraint)
Collection :- repeat(VisualElement, PropConstraint)
            | regroup(VisualElement+, PropConstraint)
```

Similar to PiCCL [34], but unlike most visualization frameworks, our grammar supports the nesting of any type of visual element—including collections—within a glyph. Glyphaestus provides three primary operators for glyph composition: *anchoring* a child visual element to a parent, *repeating* a visual element to create a collection, and *regrouping* existing visual elements into a collection. These operators enable the construction of hierarchies with arbitrary depth. Composition operators can be combined with property constraints (`PropConstraint`), which act on the visual properties of the contained marks.

This composition structure also supports remixing, allowing users to anchor previously created glyphs or collections into new designs. More generally, it enables an incremental and bi-directional construction workflow: users can either build larger hierarchies bottom-up through repetition and regrouping, or refine existing glyph hierarchies top-down by embedding new visual elements.

5.2 Visual properties

All marks have a *path* and an origin defined with respect to reference points, e.g., centre, top, and bottom. In addition, freeform marks with closed paths have a *skeleton*, which defines their length and shape, similarly to Data-Driven Guides [12], as well as their range of variation (see Figure 4B). The system automatically computes a skeleton from a closed path using its medial axis [5], although users can override it by drawing their own.

Like other visualization authoring tools, Glyphaestus supports common visual properties, including height, colour, position, angle, offset, and cardinality, that is, the number of members in a collection. However, as glyph hierarchies grow, the number of available properties increases rapidly. To manage this complexity, Glyphaestus introduces an *encapsulation* mechanism: properties only become manipulable variables once explicitly exposed by the user.

Exposed properties are reified as sliders (see Figure 5). Users can define value ranges, interpolation types, or fixed values, and later bind these variables to data. Sliders can also be used to specify constraints between properties (see Section 5.4). Beyond serving as controllers, sliders are first-class interactive objects: they can be repositioned on the canvas and assigned meaningful names.

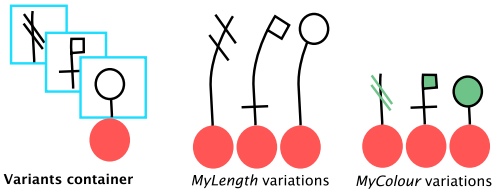


Figure 7: A container of three variants. The varying length and colour are implemented differently across variants.

5.3 Variants, types, and concepts

A distinctive feature of Glyphaestus is its support for grouping multiple *variants* of a mark (or higher-level element) to define new categorical variables. Figure 7 shows an example with three pictogram variants anchored on a circle. Users can expose additional properties of the variants, such as colour and length, and specify a variant-specific implementation of their variations. This mechanism is analogous to encapsulation and method overriding in object-oriented programming. As shown in Figure 1A, a "variant" property can be exposed and manipulated through a slider like other visual properties.

We define a *variant container* as a grouping of variants that together define a *type*. Variant containers are interactive objects: users can manipulate them on the canvas, anchor them to visual elements or collections, and update them (e.g., by adding new variants). Glyphaestus further supports typing through naming: visual elements sharing the same name are considered of the same type and can belong to the same variant container or collection.

Finally, as discussed in Section 4, Glyphaestus introduces *concepts* as a mechanism for attaching custom semantics to glyphs (or sub-glyphs) prior to data binding. A concept is an interactive object in the canvas containing a sub-glyph (or a collection) that can be linked to a data column, such that each instance of the concept corresponds to a unique value of the associated variable. For example, linking the *Country* concept to a country column of the data (see Figure 4E) causes each instance of the encapsulated glyph to represent a unique country, grouping together all rows associated with that value.

5.4 Layout and property constraints

The path, reference points, and skeleton (when applicable) of each mark act as snapping guides (points or curves) that facilitate interaction. They also serve as guides for anchoring elements and laying out collections. For example, a collection can be arranged

along the periphery of a circle (as in Figure 1) or along the skeleton of a freeform shape.

Glyphaestus provides a dedicated view for configuring layout and other collection constraints (see Figure 1D). Inspired by TimeSplines' lens view [21], it presents a linearized representation of the collection and offers interactive controls for specifying spatial constraints. Figure 6 illustrates layout configurations supported by Glyphaestus, largely based on principles of persistent alignment and distribution [6].

Constraints can be applied to any visual property of a collection, not only position. Glyphaestus supports three types of constraints: (i) equality, as in Charticulator [26], StructGraphics [36], and PiCCL [34], (ii) random, and (iii) function-of-rank (see Figure 4G). The latter enables, for example, expressing proportional relationships between visual properties, such as the sizes of the circles in the lollipop glyphs shown in Figure 2c. This approach bridges visualization authoring with procedural design systems such as Para [9].

5.5 Visualization generation

Glyphaestus focuses primarily on glyph construction rather than complete visualization authoring. As such, it currently provides limited support for global layouts and interpretive elements such as legends, while axes are only partly supported. Instead, it provides a view for displaying variations of a designed glyph on a grid (see Figure 4H), either driven by a dataset (when data bindings are defined) or by random sampling based on the glyph's parametric design, i.e., its property constraints and the ranges of exposed properties. This view enables designers to explore the space of variations and evaluate their designs.

5.6 Implementation details

Glyphaestus is implemented in C# using the Microsoft .NET framework. Freeform marks are modelled as Bézier curves. To derive their skeleton, we compute a medial axis using an implementation by Steenkamp[3] of the algorithm by Choi et al. [5]. We then place deformation handles along the skeleton and compute weights using Bounded Biharmonic Weights [10] via C#–C++ interoperability.

6 User study

We conducted a preliminary user study to evaluate whether new users can develop a functional understanding of Glyphaestus's glyph composition and variation framework, and how the interface supports (or hinders) their design workflow. The study was structured as a visualization reproduction study [27].

6.1 Method

Participants. We recruited 14 participants (5 men, 9 women) from our university campus, with diverse backgrounds in graphic design and data visualization. Three participants reported academic training in graphic design and four reported professional experience; six reported casual practice, and one reported no prior experience.

In terms of data visualization, six participants reported academic training, three reported professional experience, while two reported no prior exposure. Eight participants reported having previously designed data visualizations (four rarely, two occasionally, and two frequently). Most participants were familiar with Adobe Illustrator (9/14). Some mentioned Figma (3/14) and Canva (3/14). For data visualization, they reported using Excel (10/14), Tableau (4/14), D3 (5/14), and R or Python (7/14), with varying levels of frequency. Two participants (P7 and P13) reported professional level on both design and data visualization.

Apparatus. Participants interacted with Glyphaestus on a Dell laptop running Windows 11. They were also provided with pen and paper for sketching. We recorded screen activity and audio throughout the session.

Task and procedure. Participants first completed a background questionnaire, followed by a guided tutorial presented by the experimenter. The tutorial introduced the core concepts of Glyphaestus, including parameter exposure, design space exploration, and hierarchical composition. These concepts were illustrated through the construction of a simple grouped bar chart based on a government spending dataset. Participants were then given a dataset inspired by a visualization by Andy Kriebel (see Figure 3), consisting of "thoughts" described by four attributes: *inward/outward*, *intensity* (ordinal), *topic*, and *location*. As a preparation task, participants were asked to sketch a possible visual representation on paper. This step ensured familiarity with the data and encouraged independent reflection on potential encodings and groupings.

In the main task, participants were asked to reproduce Kriebel's visualization using Glyphaestus. This involved constructing glyphs, such as the concentric circles, exposing and adjusting parameters, such as size and colour, and matching the overall visual structure. Participants then loaded the dataset and established mappings, including grouping by the *inward/outward* attribute. Participants were further asked to extend the design by adding pictograms within collections using the *variants* construct, and by recreating stacked circle arrangements. If time permitted, they implemented an additional encoding of *intensity* using a collection of dots, inspired by designs found in Dear Data, such as the examples in Figure 2.

Sessions were conducted with a think-aloud protocol. The experimenter intervened when participants were blocked or engaged in prolonged ineffective actions.

6.2 Results

We analyzed audio transcripts alongside screen recordings, focusing on themes reflecting the system's key concepts. We report points of success, areas of confusion, and key limitations identified by participants. Study instruments and detailed data from the study are available at https://osf.io/b4re7.

Assembling marks and exposing properties. Participants understood the relationship between the glyph design inside the canvas and the global visualization, successfully grouping marks and using collections to reproduce target designs. Exposing visual variables through sliders was perceived as particularly intuitive. P13 and P14 appreciated the fact that properties become manipulable objects: "*I really liked having the slider as something persistent that you could manipulate as any object*" (P14).

However, sliders could became difficult to locate on the canvas, requiring navigation through pan and zoom. P1 and P3 suggested

[3] https://github.com/FlorisSteenkamp/MAT

improvements such as adjustable slider length for custom precision. P13 also noted that the canvas could become visually cluttered (with elements "*like green handles and red outlines*"), suggesting that some elements may not need to be constantly visible.

Graphics-first design approach. Participants appreciated the ability to design glyphs independently of data. P4 stated that "*graphics then data binding makes much more sense.*" While P7 typically starts with data, the participant explained that "*starting with graphics let me define my own shapes, not prior to linking but in parallel, going back and forth from the data to the marks.*"

Similarly, P9 and P10 valued being able to make functional designs and test variations without relying on data values: "*I needed to test the circle design to check that they [the two circles] were not overlapping, I have to do this back and forth but no data is necessary at this stage*" (P10).

Defining concepts and binding data. While participants found *Concepts* useful ("*Give meaning, it makes sense*" (P5)), several (P4, P6, P9, P11) initially struggled to understand their role, particularly in relation to collections. During this step, the experimenter intervened to assist P4 and P9. As P11 noted, "*I was confused [...] but then I got it and I like the fact that you can nest concepts, I found this very logical and useful.*" P11 also liked the possibility to make data bindings by dragging data columns to defined concepts, whereas P12 emphasized "*the flexibility to be able to go back, unlink, trying different groupings.*"

Design-iteration workflows. Participants valued the flexibility of the workflow, including the ability to revise mappings and compositions without having to restart from scratch. They explored multiple construction strategies. While most followed a bottom-up approach when creating *variant containers* by gathering pictograms already present in the canvas, others (P2, P6, P13) adopted a top-down strategy by first creating a single variant and anchoring it to a glyph, exposing is as a parameter, and later refining it by adding pictogram variations.

Some participants used variants to encode *intensity* through repeated elements (dots), instead of using collections, indicating alternative interpretations of the system's constructs. P10 highlighted the system's expressive potential: "*I foresee a lot of potential to go very far following my imagination.*" On the downside, P13 (professional designer and visualization expert) regretted the lack of support for managing parallel design alternatives, such as the use of *artboards* in tools like Adobe Illustrator or Affinity Designer.

Recurring issues. A common difficulty of all participants was reproducing the triangular layout of Kriebel's visualization (see Figure 3), as the system's version used in the study did not provide built-in support for this layout. Most participants attempted workarounds, such as drawing a zig-zag curves to serve as a guide for nested collections of glyphs representing negative thoughts; however, such strategies often did not yield the desired outcome. We discuss how we address such layout definition challenges in the following section.

7 Discussion

The study results suggest that Glyphaestus supports alternative design paths but also reveals challenges in achieving precise designs and managing layouts of nested collections. While several authoring systems provide algorithmic layouts [29], these are often hard-coded and difficult to reconcile with direct-manipulation concepts. A key challenge for systems such as Glyphaestus is therefore to support more expressive layouts while preserving their underlying interaction principles.

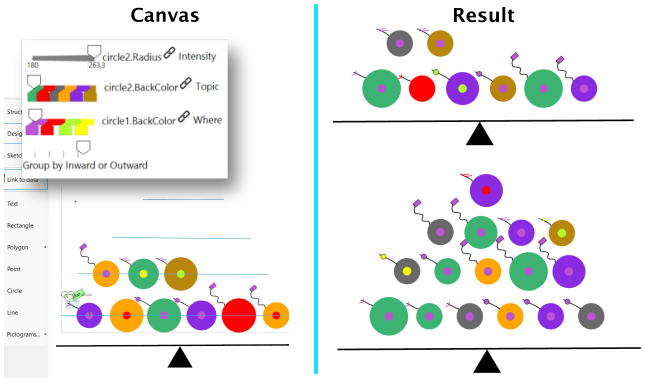


Figure 8: Reconstruction of A. Kriebel's visualization (see Figure 3), using a guide of parallel lines within a triangle.

Figure 8 presents our post-study implementation for reconstructing Kriebel's visualization (see supplementary material). Our solution is to introduce marks in the form of *discontinuous polylines*, i.e., polylines whose segments are not connected. These marks are then used as guides to distribute glyphs across multiple rows within a nested collection, using Glyphaestus's alignment and distribution mechanisms (see Figure 6). This simple yet powerful idea generalizes to discontinuous polycurves of arbitrary shapes, enabling a wide range layout configurations for nested collections.

Figure 9 presents additional examples of glyph representations created with Glyphaestus. Their step-by-step construction is provided as videos in the supplementary material. The first example (a) demonstrates heterogeneous collections anchored to the same parent (the lollipop stick), as well as functional constraints applied to circle size. This configuration is challenging to express in existing systems: even DataGarden [22], which supports tree-based structures of hand-drawn glyphs, cannot support either size constraints or multiple nested collections (circles vs. small lines), where each collection encodes a different data dimension.

The second example (b) shows nested collections with varying cardinalities (flower petals), which themselves contain sub-collections of dots. This illustrates the use of deformable shapes whose structure is not fixed within a collection: in contrast to Data-Driven Guides [12], both the number of deformable elements and the structure of child collections can vary, enabling multiple levels of flexible, data-driven variation. The third example (c) is more conventional, being reproduced by other visualization authoring systems [26, 36]. It demonstrates how Glyphaestus can still be used to express more structured visualizations as parametrized glyphs.

Lastly, Figure 1 presents a radial layout of rectangles and dot collections combined with two distinct hand-drawn marks, reproducing G. Lupi's design for Week 8 (see Figure 3b). While Charticulator [26] can support such flexible layouts through radial and free-form scaffolds, its glyph editor does not allow collections to be

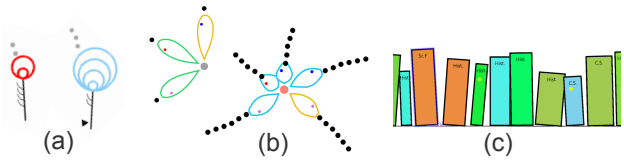


Figure 9: Examples of glyph variations reproduced using Glyphaestus: (a) Week 36 by G. Lupi, (b) Week 38 by G. Lupi, and (c) Best Bookshelf by Tanyoung Kim, where a shelf of books itself is a higher-level glyph with a nested collection.

embedded within glyphs and provides no support for combining heterogeneous visual elements along the same scaffold.

Despite these results, several limitations remain. Designs involving freeform shapes or layouts often appear sketchy or imprecise. While this may be appropriate for personal data visualization scenarios, it can hinder clarity in communicative contexts. As noted in prior work [22], integrating interpretative elements such as axes into freeform visualizations remains an open problem. Glyphaestus does not address this issue, which we identify as an important direction for future work. Finally, understanding the trade-offs of different authoring workflows and assessing the usability of the system would require a more comprehensive evaluation, for instance through a comparative study or more open-ended design tasks in which participants create their own glyph representations.

8 Conclusion

We introduced Glyphaestus, a visualization authoring system for designing parametric, drawing-based glyphs that supports hierarchical composition, explicit parametrization, and constraint-based layout. By treating glyphs as structured graphical systems, Glyphaestus enables designers to progressively move from graphical constructions to data semantics, supporting flexible and exploratory design workflows.

Our study shows that this approach facilitates alternative design paths and encourages creative exploration, while also highlighting challenges related to layout expressivity and precision. These findings point to important directions for future work, including richer layout mechanisms and improved support for balancing expressiveness with readability. More broadly, this work contributes to bridging the gap between expressive, design-driven visualization practices and formal visualization systems, and suggests a path toward more expressive, design-oriented visualization authoring.

9 Acknowledgements

This work is supported by the French National Research Agency under grant ANR-21-CE33-0002 GLACIS.

References

[1] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, polymorphism and reuse: three principles for designing visual interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Palermo, Italy) *(AVI '00)*. Association for Computing Machinery, New York, NY, USA, 102–109. doi:10.1145/345513.345267

[2] Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. 2016. Iterating between tools to create and edit visualizations. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 481–490.

[3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

[4] Matthew Brehmer, Robert Kosara, and Carmen Hull. 2021. Generative design inspiration for glyphs with Diatoms. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2021), 389–399.

[5] Hyeong In Choi, Sung Woo Choi, Hwan Pyo Moon, and Nam-Sook Wee. 1997. New algorithm for medial axis transform of plane domain. *Graphical Models and Image Processing* 59, 6 (1997), 463–483.

[6] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E Mackay, and Michel Beaudouin-Lafon. 2016. Beyond snapping: Persistent, tweakable alignment and distribution with StickyLines. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 133–144.

[7] Sonia Hashim, Tobias Höllerer, and Jennifer Jacobs. 2023. Drawing Transforms: A Unifying Interaction Primitive to Procedurally Manipulate Graphics across Style, Space, and Time. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[8] Raphaël Hoarau and Stéphane Conversy. 2012. Augmenting the scope of interactions with implicit and explicit graphical structures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1937–1946.

[9] Jennifer Jacobs, Sumit Gogia, Radomír Měch, and Joel R Brandt. 2017. Supporting expressive procedural art creation through direct manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 6330–6341.

[10] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

[11] Nam Wook Kim, Hyejin Im, Nathalie Henry Riche, Alicia Wang, Krzysztof Gajos, and Hanspeter Pfister. 2019. Dataselfie: Empowering people to design personalized visuals to represent their data. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[12] Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilmot Li, Jovan Popovic, and Hanspeter Pfister. 2016. Data-Driven Guides: Supporting Expressive Design for Information Graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 491–500.

[13] Zhicheng Liu, Chen Chen, and John Hooker. 2025. Manipulable Semantic Components: A Computational Representation of Data Visualization Scenes. *IEEE Transactions on Visualization & Computer Graphics* 31, 01 (Jan. 2025), 732–742. doi:10.1109/TVCG.2024.3456296

[14] Zhicheng Liu, Chen Chen, Francisco Morales, and Yishan Zhao. 2021. Atlas: Grammar-based Procedural Generation of Data Visualizations. In *2021 IEEE Visualization Conference (VIS)*. 171–175. doi:10.1109/VIS49827.2021.9623315

[15] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. 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*. 1–13.

[16] Giorgia Lupi and Stefanie Posavec. 2015. Dear Data. (2015). https://www.dear-data.com

[17] Wendy Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive System. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*.

[18] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E Mackay. 2017. Beyond grids: Interactive graphical substrates to structure digital layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 5053–5064.

[19] Gonzalo Gabriel Méndez, Uta Hinrichs, and Miguel A Nacenta. 2017. Bottom-up vs. top-down: Trade-offs in efficiency, understanding, freedom and creativity with infovis tools. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 841–852.

[20] Gonzalo Gabriel Méndez, Miguel A Nacenta, and Sebastien Vandenheste. 2016. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 4073–4085.

[21] Anna Offenwanger, Matthew Brehmer, Fanny Chevalier, and Theophanis Tsandilas. 2024. TimeSplines: Sketch-Based Authoring of Flexible and Idiosyncratic Timelines. *IEEE Transactions on Visualization and Computer Graphics* 30, 1 (2024), 34–44. doi:10.1109/TVCG.2023.3326520

[22] Anna Offenwanger, Theophanis Tsandilas, and Fanny Chevalier. 2025. DataGarden: Formalizing Personal Sketches into Structured Visualization Templates. *IEEE Transactions on Visualization and Computer Graphics* 31, 1 (2025), 1268–1278. doi:10.1109/TVCG.2024.3456336

[23] Josh Pollock, Catherine Mei, Grace Huang, Elliot Evans, Daniel Jackson, and Arvind Satyanarayan. 2024. Bluefish: Composing Diagrams with Declarative Relations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 23, 21 pages. doi:10.1145/3654777.3676465

[24] Josh Pollock and Arvind Satyanarayan. 2026. GoFish: A Grammar of More Graphics! *IEEE Transactions on Visualization and Computer Graphics* 32, 1 (2026), 549–559. doi:10.1109/TVCG.2025.3634250

[25] Anran Qi, Theophanis Tsandilas, Ariel Shamir, and Adrien Bousseau. 2025. Sketch2Data: Recovering data from hand-drawn infographics. *Computers &*

Graphics 130 (2025), 104251. doi:10.1016/j.cag.2025.104251

[26] Donghao Ren, Bongshin Lee, and Matthew Brehmer. 2018. Charticulator: Interactive construction of bespoke chart layouts. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 789–799.

[27] Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche. 2018. Reflecting on the evaluation of visualization authoring systems: Position paper. In *2018 IEEE Evaluation and Beyond-Methodological Approaches for Visualization (BELIV)*. IEEE, 86–92.

[28] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An interactive visualization design environment. In *Computer graphics forum*, Vol. 33. Wiley Online Library, 351–360.

[29] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2019. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 461–471.

[30] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-Lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[31] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 669–678.

[32] Toby Schachman. [n. d.]. Apparatus: A hybrid graphics editor and programming environment for creating interactive diagrams. https://aprt.us/

[33] Jeffrey A. Shaffer and Andy Kriebel. 2015. Dear Data Two. (2015). http://www.dear-data-two.com

[34] Haoyan Shi, Yunhai Wang, Junhao Chen, Chenglong Wang, and Bongshin Lee. 2025. PiCCL: Data-Driven Composition of Bespoke Pictorial Charts. *IEEE Transactions on Visualization and Computer Graphics* (2025), 1–11. doi:10.1109/TVCG.2025.3634264

[35] Xinyu Shi, Shunan Guo, Jane Hoffswell, Gromit Yeuk-Yin Chan, Victor S. Bursztyn, Jian Zhao, and Eunyee Koh. 2025. Comprehensive Sketching: Exploring Infographic Design Alternatives in Parallel. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 145, 8 pages. doi:10.1145/3706599.3720182

[36] Theophanis Tsandilas. 2020. StructGraphics: Flexible visualization design through data-agnostic and reusable graphical structures. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 315–325.

[37] Theophanis Tsandilas, Anastasia Bezerianos, and Thibaut Jacob. 2015. Sketchsliders: Sketching widgets for visual exploration on wall displays. In *Proceedings of the 33rd annual acm conference on human factors in computing systems*. 3255–3264.

[38] Yun Wang, Haidong Zhang, He Huang, Xi Chen, Qiufeng Yin, Zhitao Hou, Dongmei Zhang, Qiong Luo, and Huamin Qu. 2018. Infonice: Easy creation of information graphics. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[39] Hadley Wickham and Carson Sievert. 2009. *ggplot2: elegant graphics for data analysis*. Vol. 10. springer New York.

[40] Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-Oriented Drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 4610–4621.

[41] Haijun Xia, Bruno Araujo, and Daniel Wigdor. 2017. Collection objects: Enabling fluid formation and manipulation of aggregate selections. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 5592–5604.

[42] Haijun Xia, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor. 2018. DataInk: Direct and creative data-oriented drawing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.

[43] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: from mathematical notation to beautiful diagrams. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 144–1.

[44] Brent Yorgey and Jeffrey Rosenbluth. [n. d.]. Diagrams. https://diagrams.github.io.

Received 20 February 20xx; revised 12 March 20xx; accepted 5 June 20xx