# Topological Equivariant Artist Model

A, B, C

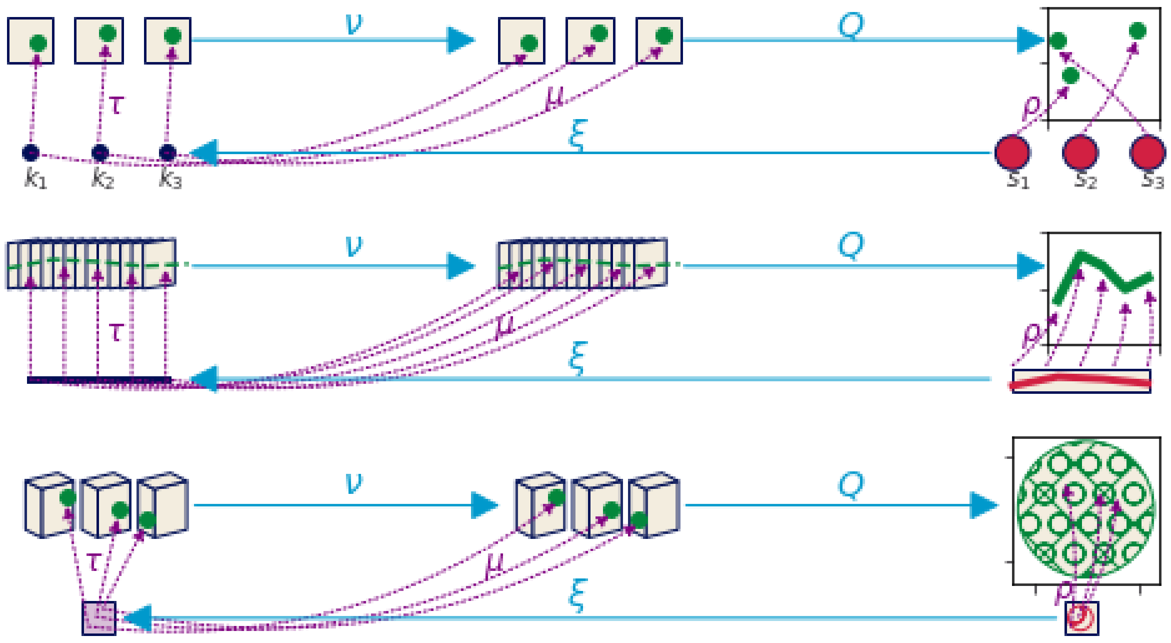


Fig. 1: Visualizations consist of topologically equivariant maps. There is a set of monoid action equivariant maps from data components to visual components ν that are then reduced via Q into a single graphic, and there is a deformation retraction ξ, from graphic continuity to data continuity.

**Abstract**— A critical aspect of data visualization is that the graphical representation of data is expected to match the properties of the data; this fails when order is not preserved in representations of ordinal data or scale for numerical data. In this work, we propose that the mathematical notions of equivariance and topology formalizes the expectation of matching properties. We developed a model we call the topological artist model (TAM) in which data and graphics can be viewed as sections of fiber bundles. This model allows for (1) decomposing the translation of data fields (variables) into visual channels via an equivariant map on the fibers and (2) a topology-preserving map of the base spaces that translates the dataset connectivity into graphical elements. Furthermore, our model supports an algebraic sum operation such that more complex visualizations can be built from simple ones. We illustrate the application of the model through case studies of a scatter plot, line plot, and heatmap. We show that this model can be implemented with a small prototype.

**Index Terms**—Taxonomy, Models, Frameworks, Theory

## 1 Introduction

The aim of this work is to rearchitecture Matplotlib to take advantage of developments in software design, data structures, and visualization to improve consistency, reusability, and discoverability, so domain specific tool developers can build structure preserving visualization tools. The contribution of this work is (a) topology preserving relationship between data and graphic via continuous maps, (b) property preservation from data component to visual representation as equivariant maps that carry a homomorphism of monoid actions, (c) functional oriented visualization tool architecture built on the mathematical model to demonstrate the utility of the model, (d) prototype of the architecture built on Matplotlib's infrastructure to demonstrate the feasibility of the model.

- *A B are with the Computer Science department, City College of New York. E-mail: roy.g.biv@aol.com.*
- *Thomas Caswell is with Brookhaven National Lab E-mail: ed.grimley@aol.com.*

*Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx*

## 2 Related Work

The underlying structure and semantics of visualization are by definition reflected in visual representations [30], whether through direct mappings from data into visual elements or via figurative representations that have meaning due to their similarity to external concepts [22]. The components of a visual representation were first codified by Bertin [14], and the notion that the properties of the data and visual representation match is the basis of most evaluations of visualization. Expressiveness, as defined by Mackinlay [45, 46] is a measure of how much of the structure any map can encode. A fully expressive component is one that is equivariant since it has preserved all the structure in the data. Models of visualization evaluate this equivariance and how elements built in the model can be composed to build more complex visualizations. Mackinlay's A Presentation Tool introduced the notion of visualizations having syntax and semantics [45] and Wilkenson described the grammar of this language [67]. This grammar oriented approach allows users to describe how to compose visual elements into a graphical design [69], while we are proposing a framework for building those elements. This same limitation is explicitly stated in the functional dependency model of visualization developed by Sugibuchi [62] and evident in Vickers' category theory oriented framework in which semiotics are commutative. The algebraic process model by Kindlmann and Scheideggar proposes that the data and visualization transformations are commutative; while similar to our framework, it is missing any explicit mention of continuity.

On the other hand, most information visualization tools are explicitly tuned to specific continuity [38, 63]. For example, the relational database is core to libraries built on top of Grammar of Graphics [67], including ggplot [65], protovis [16] and D3 [17], vega [53] and altair [64]. Images underpin scientific visualization tools such as Napari [55] and ImageJ [54] and the digital humanaties oriented ImagePlot [61] macro. Neither the table nor image model on its own supports all the data types a typical general purpose visualization library needs to support; instead libraries such as Matplotlib [41] and Vtk [32, 37] explicitly carry around different data representations for all the different types of visualizations they support. Where libraries with a single core data structure have very consistent APIs, VTK and Matplotlib APIs can be rather inconsistent as every visualization has a different notion of how the data is structured.

Fiber bundles are a generalized abstraction proposed by Butler to encode the continuity of the data separately from the variable information [20, 21]. Since Butler's model lacks a robust way of describing variables, we fold in Spivak's Simplicial formulation of databases [57, 58] to incorporate a schema like description of the variables. One way of describing the binding between the schame and the continuinity is using the notion of structural *keys* with associated *values* proposed by Munzner [48]. Unlike Munzner's model where the semantic meaning of the key is tightly coupled to the index of the value, our model considers keys to be a reference to topology. This allows the metadata to be altered, for example by changing the coordinate system or time resolution, without imposing new semantics on the underlying structure.

## 3 Topological Equivariant Artist Model

We introduce the notion of an artist $\mathscr{A}$ as an equivariant map from data to graphic

$$\mathscr{A} : \mathscr{E} \to \mathscr{H} \tag{1}$$

that carries a homomorphism of monoid actions $\varphi : M \to M'$ [26], which are discussed in detail in Sect. 3.1.2. Given $M$ on data $\mathscr{E}$ and $M'$ on graphic $\mathscr{H}$, we propose that artists $\mathscr{A}$ are equivariant maps

$$\mathscr{A}(m \cdot r) = \varphi(m) \cdot \mathscr{A}(r) \tag{2}$$

such that applying a monoid action $m \in M$ to the data $r \in \mathscr{E}$ input to $\mathscr{A}$ is equivalent to applying a monoid action $\varphi(M) \in M'$ to the graphic $A(r) \in \mathscr{H}$ output of the artist. We model the data $\mathscr{E}$, graphic $\mathscr{H}$, and intermediate visual encoding $\mathscr{V}$ stages of visualization as topological structures that encapsulate types of variables and continuity; by doing so we can develop implementations that keep track of both in ways that let us distribute computation while still allowing assembly and dynamic update of the graphic.

### 3.1 Data Bundle

Building on Butler's proposal of using fiber bundles as a common data representation structure for visualization data [20, 21], a fiber bundle is a tuple $(E, K, \pi, F)$ defined by the projection map $\pi$

$$\begin{tikzcd} F \arrow[r, hook] & E \arrow[r, "\pi"] & K \end{tikzcd} \tag{3}$$

that binds the components of the data in $F$ to the continuity represented in $K$. By definition fiber bundles are locally trivial [3, 56], meaning that over a localized neighborhood we can dispense with extra structure on $E$ and focus on the components and continuity.

#### 3.1.1 Fiber Space: Variables

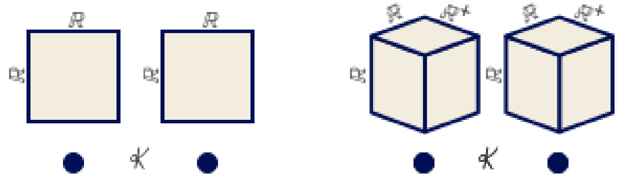


Fig. 2: These two datasets have the same base space $K$ of discrete points, but the square is a visual representation of the fiber space $F = \mathbb{R} \times \mathbb{R}$, while the cube is a representation of $F = \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R}$

To formalize the structure of the data components, we use notation introduced by Spivak [57] that binds the components of the fiber to variable names. Spivak constructs a set $\mathbb{U}$ that is the disjoint union of all possible objects of types $\{T_0, \ldots, T_m\} \in \mathbf{DT}$, where $\mathbf{DT}$ are the data types of the variables in the dataset. He then defines the single variable set $\mathbb{U}_\sigma$

$$\begin{tikzcd} \mathbb{U}_\sigma \arrow[r] \arrow[d, "\pi_\sigma"'] & \mathbb{U} \arrow[d, "\pi"] \\ C \arrow[r, "\sigma"'] & \mathbf{DT} \end{tikzcd} \tag{4}$$

which is $\mathbb{U}$ restricted to objects of type $T$ bound to variable name $c$. The $\mathbb{U}_\sigma$ lookup is by name to specify that every component is distinct, since multiple components can have the same type $T$. Given $\sigma$, the fiber for a one variable dataset is

$$F = \mathbb{U}_{\sigma(c)} = \mathbb{U}_T \tag{5}$$

where $\sigma$ is the schema binding variable name $c$ to its datatype $T$. A dataset with multiple variables has a fiber that is the cartesian cross product of $\mathbb{U}_\sigma$ applied to all the columns:

$$F = \mathbb{U}_{\sigma(c_1)} \times \ldots \mathbb{U}_{\sigma(c_i)} \ldots \times \mathbb{U}_{\sigma(c_n)} \tag{6}$$

which is equivalent to

$$F = F_0 \times \ldots \times F_i \times \ldots \times F_n \tag{7}$$

which allows us to decouple $F$ into components $F_i$. Each component of $F$ is a dimension of the topological fiber space, as illustrated in Fig. 2. The space $F = \mathbb{R} \times \mathbb{R}$ encodes that the dataset has two quantative continuous variables, while $F = \mathbb{R} \times \mathbb{R}\mathbb{R}^+$ encodes a dataset with three quantative contunous variables, one of which must be positive.

### 3.1.2 Equivariant Variable Properties: Monoid Actions

While structure on a set of values is often described algebraically as operations or through the actions of a group, for example Steven's measurement scales [43,59], we generalize to monoids to support partial orderings. A partial ordering allows for multiple measurement values to have the same rank [29], which is useful for visualizing many types of multi indicator systems [18].

A monoid [8] $M$ is a set with an associative binary operator $*: M \times M \to M$. A monoid has an identity element $e \in M$ such that $e * a = a * e = a$ for all $a \in M$. As defined on a component of $F$, a left monoid action [1,9] of $M_i$ is a set $F_i$ with an action $\bullet : M \times F_i \to F_i$ with the properties:

**associativity** for all $f, g \in M_i$ and $x \in F_i$, $f \bullet (g \bullet x) = (f * g) \bullet x$

**identity** for all $x \in F_i, e \in M_i$, $e \bullet x = x$

As with the fiber $F$ the total monoid space $M$ is the cartesian product

$$M = M_0 \times \ldots \times M_i \times \ldots \times \ldots M_n \tag{8}$$

of each monoid $M_i$ on $F_i$. The monoid is also added to the specification of the fiber $(c_i, T_i, \mathbb{U}_\sigma M_i)$

Defining the monoid actions on the components serves as the basis for identifying the invariance [42] that must be preserved in the visual representation of the component. A secondary advantage of defining structure in terms of monoids is that they are commonly found in functional programming because they specify compositions of transformations [60,70].

### 3.1.3 Base Space: Continuity

The base space $K$ is way to express how the records in $E$ are connected to each other, for example if they are discrete points or if they lie in a 2D continous surface. Formally $K$ is the quotient space of $E$ meaning it is the finest space [11] such that every $k \in K$ has a corresponding fiber $F_k$ [6]. Our model formulates $K$ as akin to an indexing space into $E$ that describes the structure of $E$.

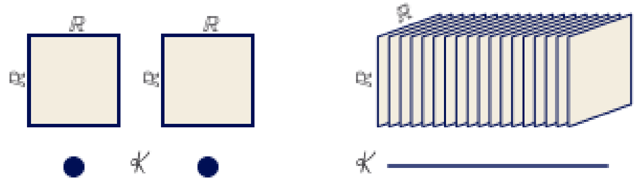


Fig. 3: These two datasets have the same fiber space $F = \mathbb{R} \times \mathbb{R}$, but the left dataset has a discrete continuity while the right is 1D continuous over the interval $[0,1]$.

As illustrated in Fig. 3, $K$ can have any number of dimensions, can be continuous or discrete, and is somewhat independent of the dimensions of the fiber. As with fibers and monoids, we can decompose the total space into components $\pi : E_i \to K$ where

$$\pi : E_1 \oplus \ldots \oplus E_i \oplus \ldots \oplus E_n \to K \tag{9}$$

which is a decomposition of $F$. The $K$ remains the same because the connectivity of records does not change just because there are fewer elements in each record. By encoding this continuity in the model as $K$ the data model now explicitly carries information about its structure such that the implicit assumptions of the visualization algorithms are now explicit. The explicit topology is a concise way of distinguishing visualizations that appear identical, for example heatmaps and images.

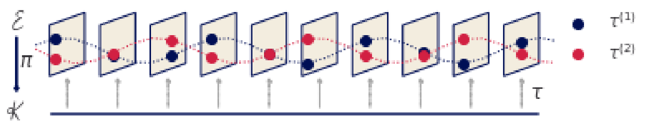


Fig. 4: Each section in the fiber bundle is a unique continuous map from base space to fiber encoding the set of records in the dataset. The two sections, $\tau^{(1)}$ and $\tau^{(2)}$, shown here are in the set of global sections $\Gamma(E)$.

### 3.1.4 Section: Values

While the projection function $\pi : E \to K$ ties together the base space $K$ with the fiber $F$, a section $\tau : K \to E$ encodes a dataset. A section function takes as input location $k \in K$ and returns a record $r \in E$. For any fiber bundle, there exists a map

$$\begin{tikzcd} F \arrow[r, hook] & E \arrow[d, shift right, "\pi"'] \\ & K \arrow[u, bend right, "\tau"'] \end{tikzcd} \tag{10}$$

such that $\pi(\tau(k)) = k$. The set of all global sections is denoted as $\Gamma(E)$. As illustrated in Fig. 4, the section is a continuous mapping from a location $k \in K$ on the base space to a record $r \in F$ in the fiber. Assuming a trivial fiber bundle $E = K \times F$, the section is

$$\tau(k) = (k, (g_{F_0}(k), \ldots, g_{F_n}(k))) \tag{11}$$

where $g : K \to F$ is the index function into the fiber component. This formulation of the section also holds on locally trivial sections of a non-trivial fiber bundle. As with Equation 7 and Equation 9, $\tau$ can be decomposed into components

$$\tau = (\tau_0, \ldots, \tau_i, \ldots, \tau_n) \tag{12}$$

where each section $\tau_i$ maps into a record on a component $F_i \in F$. This allows for accessing the data component wise in addition to accessing the data in terms of its location over $K$.

### 3.1.5 Sheafs: Gluing Sections

The restriction maps of a sheaf describe how local sections $\iota^*\tau$ over a neighborhood $U$ around $k$ can be glued into larger sections [33, 34]. This puts a continuous structure on local sections, which allows for defining a section over a subset in $K$. The inclusion map $\iota : U \to K$ pulls $E$ over $U$

$$\begin{tikzcd} \iota^*E \arrow[r, hook, "\iota^*"] \arrow[d, shift right, "\pi"'] & E \arrow[d, shift right, "\pi"'] \\ U \arrow[r, hook, "\iota"] \arrow[u, bend right, "\iota^*\tau"'] & K \arrow[u, bend right, "\tau"'] \end{tikzcd} \tag{13}$$

such that the pulled back $\iota^*\tau$ only contains records over $U \subset K$. Gluing together $\iota^*\tau$ sections is necessary for navigation techniques such as pan and zoom [50] and dynamically updated visualizations such as sliding windows [27,28].

## 3.2 Graphic Bundle

We introduce a graphic bundle to hold the essential information necessary to render a graphical design constructed by the artist. As with the data, we can represent the target graphic as a section $\rho$ of a bundle $(H, S, \pi, D)$

$$\begin{tikzcd} D \arrow[r, hook] & H \arrow[d, shift right, "\pi"'] \\ & S \arrow[u, bend right, "\rho"'] \end{tikzcd} \tag{14}$$

where $\rho$ is the fully encoded graphic. To fully specify the visual characteristics of the image, we construct a fiber D that is an infinite resolution version of the target space. Typically H is trivial and therefore sections can be thought of as mappings into D. In this work, we assume a 2D opaque image $D = \mathbb{R}^5$ with elements $(x, y, r, g, b) \in D$ such that a rendered graphic only consists of 2D position and color. By abstracting the target display space as D, the model can support different targets, such as a 2D screen or 3D printer.

### 3.2.1 Equivariant Topology: Graphic Base Space

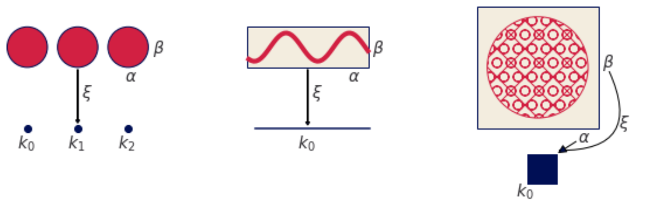


Fig. 5: The 0D scatter k and 1D line k are thickened into S with coordinates $s = (\alpha, \beta)$ that are a region in an idealized 2D screen. The image has the same dimension in S as in K.

Just as the K encodes the connectivity of the records in the data, we propose an equivalent S that encodes the connectivity of the rendered elements of the graphic. Formally, we require that K be a deformation retract [7] of S so that K and S have the same homotopy. The surjective map $\xi : S \to K$

$$\begin{tikzcd} E \arrow[d, "\pi"'] & H \arrow[d, "\pi"'] \\ K & S \arrow[l, "\xi"'] \end{tikzcd} \tag{15}$$

goes from region $s \in S_k$ to its associated point s. While S must have the same continuity as K it is sometimes the thickened version shown in Fig. 5. This thickening is necessary when the dimensionality of K is less than the dimensionality of the target display. For example, a k that is 0D in K cannot be represented on screen unless it is thickened to 2D to encode the connectivity of the points in D that visually represent the record at k.

## 3.3 Artist

The topological artist A is a monoid equivariant sheaf map from the sheaf on a data bundle E which is $\mathcal{O}(E)$ to the sheaf on the graphic bundle H, $\mathcal{O}(H)$.

$$A : \mathcal{O}(E) \to \mathcal{O}(H) \tag{16}$$

While A can usually construct graphical elements solely with the data in $\tau$, some visualizations, such as line, may also need some finite number n of derivatives, which is captured by the jet bundle $\mathcal{J}^n$ [5,49] with $\mathcal{J}^0(E) = E$. In this work, we at most need $\mathcal{J}^2(E)$ which is the value at $\tau$ and its first and second derivatives; therefore the artist takes as input the jet bundle $E' = \mathcal{J}^2(E)$.

Specifically, Ais the equivariant map from $E'$ to a specific graphic $\rho \in \Gamma(H)$

$$\begin{tikzcd} E' \arrow[r, "\nu"] \arrow[dr, "\pi"'] & V \arrow[d, "\pi"] & \xi^*V \arrow[l, "\xi^*"'] \arrow[r, "Q"] \arrow[d, "\xi^*\pi"'] & H \arrow[dl, "\pi"] \\ & K & S \arrow[l, "\xi"'] & \end{tikzcd} \tag{17}$$

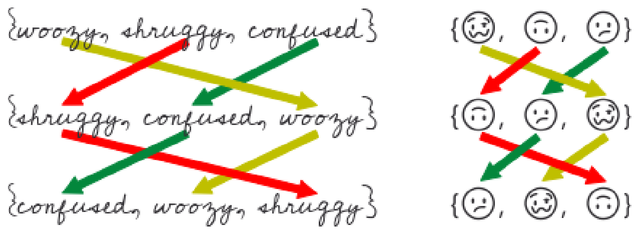
where the input can be point wise $\tau(k) \mid k \in K$. The visual bundle $(V, K, \pi, P)$ is the latent space of possible parameters of a visualization type, such as a scatter or line plot. As with the data and graphic bundles, the visual bundle is defined by the projection map $\pi$

$$\begin{tikzcd} P \arrow[r, hook] & V \arrow[d, "\pi"', shift right] \\ & K \arrow[u, "\mu"', bend right, shift right] \end{tikzcd} \tag{18}$$

where $\mu$ is the visual variable encoding [14] of the data section $\tau$. The visual fiber P is defined in terms of the input parameters of the visualization library's plotting functions; by making these parameters explicit components of the fiber, we can build consistent definitions and expectations of how these parameters behave.

In Equation 17, the encoders $\nu : E' \to V$ convert the data components to visual components. The continuity map $\xi : S \to K$ then pulls back the visual bundle V over S. Then the assembly function $Q : \xi^*V \to H$ composites the fiber components of $\xi^*V$ into a graphic in H. This functional decomposition of the visualization artist facilitates building reusable components at each stage of the transformation because the equivariance constraints are defined on $\nu$, Q, and $\xi$. We name this map the artist as that is the analogous part of the Matplotlib [40] architecture that builds visual elements.

### 3.3.1 Visual Component Maps



{woozy, shruggy, confused}
{shruggy, confused, woozy}
{confused, woozy, shruggy}

Fig. 6: In this artist, $\nu$ maps the strings to the emojis. This $\nu$ is equivariant because the monoid actions, represented by the colored arrows, are the same on the input $\tau$ and output $\mu$ sets.

We define the visual transformers $\nu$

$$\{\nu_0, \ldots, \nu_n\} : \{\tau_0, \ldots, \tau_n\} \mapsto \{\mu_0, \ldots, \mu_n\} \tag{19}$$

as the set of equivariant maps $\nu_i : \tau_i \mapsto \mu_i$. Given $M_i$ is the monoid action on $E_i$ and that there is a monoid $M_i{}'$ on $V_i$, then there is a monoid homomorphism from $\varphi : M_i \to M_i{}'$ that $\nu$ must preserve. As mentioned in Sect. 3.1.2, monoid actions define the structure on the fiber components and are therefore the basis for equivariance. A validly constructed $\nu$ is one where the diagram of the monoid transform m commutes such that

$$\begin{tikzcd} E_i \arrow[r, "\nu_i"] \arrow[d, "m_r"'] & V_i \arrow[d, "m_v"] \\ E_i \arrow[r, "\nu_i"] & V_i \end{tikzcd} \tag{20}$$

In general, the data fiber $F_i$ cannot be assumed to be of the same type as the visual fiber $P_i$ and the actions of M on $F_i$ cannot be assumed to be the same as the actions of $M'$ on P; therefore an equivariant $\nu_i$ must satisfy the constraint

$$\nu_i(m_r(E_i)) = \varphi(m_r)(\nu_i(E_i)) \tag{21}$$

such that $\varphi$ maps a monoid action on data to a monoid action on visual elements. However, we can construct a monoid action of $M$ on $P_i$ that is compatible with a monoid action of $M$ on $F_i$. We can compose the monoid actions on the visual fiber $M' \times P_i \to P_i$ with the homomorphism $\varphi$ that takes $M$ to $M'$. This allows us to define a monoid action on $P$ of $M$ that is $(m, \nu) \to \varphi(m) \bullet \nu$. Therefore, without a loss of generality, we can assume that an action of $M$ acts on $F_i$ and on $P_i$ compatibly such that $\varphi$ is the identity function.

The $\nu$ illustrated in Fig. 6 is an equivariant mapping from **Strings** to emojis, as seen in the identical actions, represented as arrows, on the data and emojis. More generally, for the Steven's scales [59] an equivariant $\nu$ has the following properties

| | | |
|---|---|---|
| nominal | permutation | if $r_1 \neq r_2$ then $\nu(r_1) \neq \nu(r_2)$ |
| ordinal | monotonic | if $r_1 \leqslant r_2$ then $\nu(r_1) \leqslant \nu(r_2)$ |
| interval | translation | $\nu(x+c) = \nu(x)+c$ |
| ratio | scaling | $\nu(xc) = \nu(x)*c$ |

where the middle column is the group structure. Therefore, a $\nu$ is equivariant if it satisfies the listed condition. For example, given a transform $\nu_i(x) = .5$ on interval data, it must commute under translation. Testing this constraint with a translation action $t(x) = x+2$

$$\begin{aligned} \nu(t(r+2)) &\overset{?}{=} \nu(r)+2 \\ .5 &\neq .5+2 \end{aligned}$$

$\nu$ does not commute and is therefore invalid. The constraints on $\nu$ can be embedded into the artist such that the $\nu$ functions can test for equivariance and also provide guidance on constructing new $\nu$ functions.

### 3.3.2 Visualization Assembly

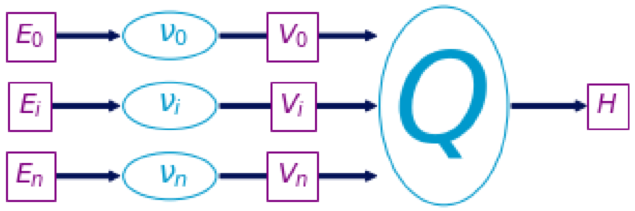


Fig. 7: $\nu_i$ functions convert data $\tau_i$ to visual characteristics $\mu_i$, then Q assembles $\mu_i$ into a graphic $\rho$ such that there is a map $\xi$ preserving the continuity of the data.

As shown in Fig. 7, $\nu$ and Q are a map-reduce operation: map the data into their visual encodings $\mu$ reduce the encodings into a graphic $\rho$. While it may seem intuitive that visualizations that generate the same graphic should do so consistently given the same input, we formalize this constraint such that it can be specified as a constraint of Q. We define the equivariant map as $Q : \mu \mapsto \rho$ and an action on the subset of graphics $Q(\Gamma(V)) \in \Gamma(H)$ that Q can generate. We then define the constraint on Qsuch that if Q is applied to two visual sections $\mu, \mu'$ that generate the same $\rho$ then the output of $\mu, \mu'$ acted on by the same monoid $m$ must be the same.

Lets call the visual representations of the components $\Gamma(V) = X$ and the graphic $Q(\Gamma(V)) = Y$. If for elements of the monoid $m \in M$ and for all $\mu, \mu' \in X$, we define the monoid action on X so that it is by definition equivariant

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \qquad (22)$$

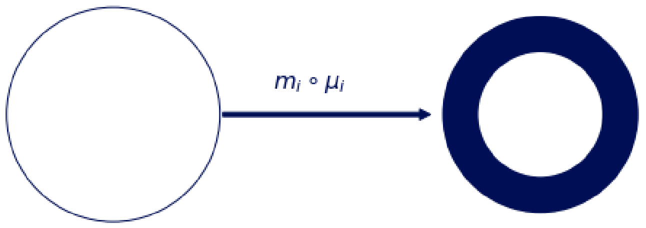


Fig. 8: These two glyphs are generated by the same Q function. The monoid action $m_i$ on edge thickness $\mu_i$ of the first glyph yields the thicker edge $\mu_i{}'$ in the second glyph.

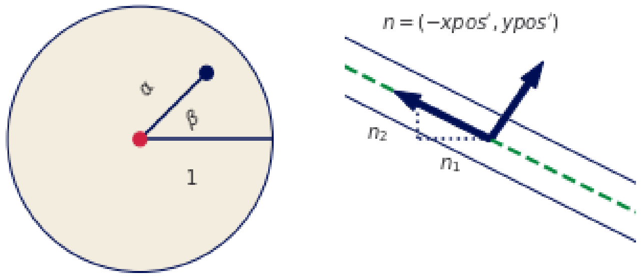


Fig. 9: The coordinates $(\alpha, \beta)$ dictate the color of the region in prerender space S which over the whole disk generates the graphical mark. The line fiber is thickened with the derivative because the tangent the line needs to be pushed perpendicular to the tangent of (xpos, ypos) in order to have visible thickness.

then a monoid action on Y can be defined as $m \circ \rho = \rho'$. The transformed graphic $\rho'$ is equivariant to a transform on the visual bundle $\rho' = Q(m \circ \mu)$ on a section that $\mu \in Q^{-1}(\rho)$ that must be part of generating $\rho$. Given fiber $P = (\mathrm{xpos}, \mathrm{ypos}, \mathrm{color}, \mathrm{thickness})$, then sections $\mu = (0,0,0,1)$ and $Q(\mu) = \rho$ generates a piece of the thin hollow circle. The action $m = (e, e, e, x+2)$, where e is identity, translates $\mu$ to $\mu' = (e, e, e, 3)$ and the corresponding action on $\rho$ causes $Q(\mu')$ to be the thicker circle in Fig. 8.

We formally describe a glyph as Q applied to the regions k that map back to a set of path connected components $J \subset K$ as input

$$J = \{j \in K \text{ s. t. } \exists \gamma \text{ s.t. } \gamma(0) = k \text{ and } \gamma(1) = j\} \qquad (23)$$

where the path [4] $\gamma$ from k to j is a continuous function from the interval [0,1]. We define the glyph as the graphic generated by $Q(S_j)$

$$\begin{tikzcd} H \arrow[r, shift left] & S_j \arrow[l, shift left, "\rho(S_j)"] \arrow[r, shift left, "\xi(s)"] & J_k \arrow[l, shift left, "\xi^{-1}(J)"] \end{tikzcd} \qquad (24)$$

such that for every glyph there is at least one corresponding region on K, in keeping with the definition of glyph as any differentiable element put forth by Ziemkiewicz and Kosara [71]. The primitive point, line, and area marks [14, 24] are specially cased glyphs.

In Fig. 1, we illustrate the output of a minimal Q that will generate distinguishable graphical marks: non-overlapping scatter points, a non-infinitely thin line, and an image. The scatter plot can be defined as $Q(\mathrm{xpos}, \mathrm{ypos})(\alpha, \beta)$ where color $\rho_{RGB} = (0,0,0)$ is defined as part of Q and $s = (\alpha, \beta)$ defines the region on S. The position of this swatch of color can be computed relative to the location on the disc $S_k$ as shown in Fig. 9

$$x = size * \alpha\cos(\beta) + xpos$$
$$y = size * \alpha\sin(\beta) + ypos$$

such that $\rho(s) = (x, y, 0, 0, 0)$ colors the point (x,y) black. In contrast, the line plot $Q(xpos, \hat{n}_1, ypos, \hat{n}_2)(\alpha, \beta)$ in Fig. 1 has a $\xi$ function that is not only parameterized on k but also on the $\alpha$ distance along k and corresponding region in SÅs shown in **??** line needs to know the tangent of the data to draw an envelope above and below each (xpos,ypos) such that the line appears to have a thickness. The magnitude of the slope is $|n| = \sqrt{n_1{}^2 + n_2{}^2}$ such that the normal is $\hat{n}_1 = \frac{n_1}{|n|}$, $\hat{n}_2 = \frac{n_2}{|n|}$ which yields components of $\rho$

$$x = xpos(\xi(\alpha)) + width * \beta\hat{n}_1(\xi(\alpha))$$
$$y = ypos(\xi(\alpha)) + width * \beta\hat{n}_2(\xi(\alpha))$$

where (x,y) look up the position $\xi(\alpha)$ on the data and the derivatives $\hat{n}_1, \hat{n}_2$ . The derivatives are then multiplied by a $width$ parameter to specify the thickness.

The image $Q(xpos, ypos, color)$ in Fig. 1 is a direct lookup into $\xi : S \to K$. The indexing variables $(\alpha, \beta)$ define the distance along the space, which is then used by $\xi$ to map into K to lookup the color values

$$R = R(\xi(\alpha, \beta)),\ G = G(\xi(\alpha, \beta)),\ B = B(\xi(\alpha, \beta))$$

In the case of an image, the indexing mapper $\xi$ may do some translating to a convention expected by Q, for example reorientng the array such that the first row in the data is at the bottom of the graphic.

### 3.3.3 Assembly Factory

The graphic base space S is not accessible in many architectures, including Matplotlib; instead we can construct a factory function $\hat{Q}$ over K that can build a Q. As shown in eq 17, Q is a bundle map $Q : \xi^*V \to H$ where $\xi^*V$ and H are both bundles over S.

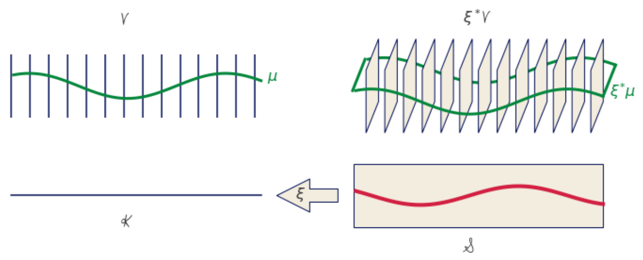


Fig. 10: Because the pullback of the visual bundle $\xi^*V$ is the replication of a $\mu$ over all points s that map back to a single k, we can constract a $\hat{Q}$ on $\mu$ over k that will fabricate the Q for the equivalent region of s associated to that k

The preimage of the continuity map $\xi^{-1}(k) \subset S$ is such that many graphic continuity points $s \in S_K$ go to one data continuity point k; therefore, by definition the pull back of $\mu$

$$\xi^*V\mid_{\xi^{-1}(k)} = \xi^{-1}(k) \times P \qquad (25)$$

copies the visual fiber P over the the points s in graphic space S that correspond to one k in data space K. This set of points s are the preimage $\xi^{-1}(k)$ of k.

As shown in Fig. 10, given the section $\xi^*\mu$ pulled back from $\mu$ and the point $s \in \xi^{-1}(k)$, there is a direct map $(k, \mu(k)) \mapsto (s, \xi^*\mu(s))$ from $\mu$ over k to the section $\xi^*\mu$ over s. This means that the pulled back section $\xi^*\mu(s) = \xi^*(\mu(k))$ is the section $\mu$ copied over all ssuch that $\xi^*\mu$ is identical for all s where $\xi(s) = k$. In Fig. 10 each dot on Pis equivalent to the line on $P^*\mu$.

Given the equivalence between $\mu$ and $\xi^*\mu$ defined above, the reliance on S can be factored out. When Q maps visual sections into graphics $Q : \Gamma(\xi^*V) \to \Gamma(H)$, if we restrict Q input to $\xi^*\mu$ then the graphic section $\rho$ evaluated on a visual region s

$$\rho(s) := Q(\xi^*\mu)(s) \qquad (26)$$

is defined as the assembly function Q with input $\xi^*\mu$ evaluated on s. Since the pulled back section $\xi^*\mu$ is the section $\mu$ copied over every graphic region $s \in \xi^{-1}(k)$, we can define a Q factory function

$$\hat{Q}(\mu(k))(s) := Q((\xi^*\mu)(s)) \qquad (27)$$

where $\hat{Q}$ with input $\mu$ is defined to Q that takes as input the copied section $\xi^*\mu$ such that both functions are evaluated over the same location $\xi^{-1}(k) = s$ in the base space S.

Factoring out s from equation 27 yields $\hat{Q}(\mu(k)) = Q(\xi^*\mu)$ where Q is no longer bound to input but $\hat{Q}$ is still defined in terms of K. In fact, $\hat{Q}$ is a map from visual space to graphic space $\hat{Q} : \Gamma(V) \to \Gamma(H)$ locally over k such that it can be evaluated on a single visual record $\hat{Q} : \Gamma(V_k) \to \Gamma(H\mid_{\xi^{-1}(k)})$. This allows us to construct a $\hat{Q}$ that only depends on K, such that for each $\mu(k)$ there is part of $\rho\mid_{\xi^{-1}(k)}$. The construction of $\hat{Q}$ allows us to retain the functional map reduce benefits of Q without having to majorly restructure the existing pipeline for libraries that delegate the construction of $\rho$ to a back end such as Matplotlib.

### 3.3.4 Composite and Reusable Artists

Given the family of artists $(E_i : i \in I)$ on the same image, the + operator

$$+ := \bigsqcup_{i \in I} E_i \qquad (28)$$

defines a simple composition of artists. When artists share a base space $K_2 \hookrightarrow K_1$, a composition operator can be defined such that the artists are acting on different components of the same section. This type of composition is important for visualizations where elements update together in a consistent way, such as multiple views [10,51] and brush-linked views [13,19]. It is impractical to implement an artist for every single graphic; instead we implement an approximation of the equivalence class of artists $\{A \in A' : A_1 \equiv A_2\}$. Roughly, two artists are equivalent if they have the same visual fiber P assembly function Q and continuity map $\xi$.

## 4 Prototype

To build a prototype, we make use of the Matplotlib figure and axes artists [40,41] so that we can initially focus on the data to graphic transformations and exploit the Matplotlib transform stack to transform data coordinates into screen coordinates.

```
fig, ax = plt.subplots()
artist = Artist(data, transforms)
ax.add_artist(artist)
```

Building on the current Matplotlib artists which construct an internal representation of the graphic, `ArtistClass` acts as an equivalance class artist $A'$. The visual bundle V is specified as the `transform`dictionary of the form `{parameter:(variable, encoder)}` where parameter is a component in P, variable is a component in F, and the $\nu$ encoders are passed in as functions or callable objects. The data bundle E is passed in as a `data` object. By binding data and transforms to $A'$ inside `__init__`, the `draw` method is a fully specified artist A.

```python
class ArtistClass(matplotlib.artist.Artist):
    def __init__(self, data, transforms, *args, **kwargs):
        # properties that are specific to the graphic
        self.data = data
        self.transforms = transforms
        super().__init__(*args, **kwargs)

    def assemble(self, **args):
        # set the properties of the graphic

    def draw(self, renderer):
        # returns K, indexed on fiber then key
        view = self.data.view(self.axes)
        # visual channel encoding applied fiberwise
        visual = {p: t['encoder'](view[t['name']])
                  for p, t in self.transforms.items()}
        self.assemble(**visual)
        # pass configurations off to the renderer
        super().draw(renderer)
```

The data is fetched in section $\tau$ via a `view` method on the data because the input to the artist is a section on $E$. The `view` method takes the `axes` attribute because it provides the region in graphic coordinates $S$ that can be used to query back into data to select a subset. To ensure the integrity of the section, `view` must be atomic, which means that the values cannot change after the method is called in draw until a new call in draw. We put this constraint on the return of the `view` method so that we do not risk race conditions.

The $\nu$ functions are then applied to the data to generate the visual section $\mu$ that here is the object `visual`. The conversion from data to visual space is simplified here to directly show that it is the encoding $\nu$ applied to the component. The `assemble` function that is $\hat{Q}$is responsible for generating a representation such that it could be serialized to recreate a static version of the graphic. This artist is not optimized because we prioritized demonstrating the separability of $\nu$ and $\hat{Q}$. The last step in the artist function is handing itself off to the renderer. The extra `*arg, **kwargs` arguments in `__init__`,`draw` are artifacts of how these objects are currently implemented.

## 4.1 Scatter and Line Artists

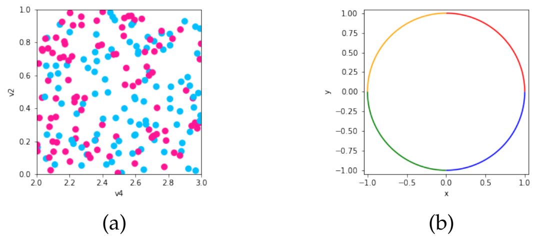


Fig. 11: Scatter plot and line plot implemented using prototype artists and data models, building on Matplotlib rendering.

To generate the figure in Fig. 11a, the `Point` artist builds on `collection` artists because collections are optimized to efficiently draw a sequence of primitive point and area marks. In this prototype, the scatter marker shape is fixed as a circle, and the only visual fiber components are x and y position, size, and the facecolor of the marker. We only show the `assemble` function here because the `__init__, draw` are identical the prototype artist. The `view` method repackages the data as a fiber component indexed table of vertices. Even though the `view` is fiber indexed, each vertex at an index $k$has corresponding values in section $\tau(k_i)$. This means that all the data on one vertex maps to one glyph.

```python
class Point(mcollections.Collection):
  def assemble(self, x, y, s, facecolors='C0' ):
    # construct geometries of circle glyphs
    self._paths = [mpath.Path.circle((xi,yi), radius=si)
                  for (xi, yi, si) in zip(x, y, s)]
    # set attributes of glyphs, these are vectorized
    # circles and facecolors are lists of the same size
    self.set_facecolors(facecolors)
```

In `assemble`, the $\mu$ components are used to construct the vector path of each circular marker with center $(\mathtt{x},\mathtt{y})$ and size `x` and set the colors of each circle. This is done via the `Path.circle` object.

```python
class Line(mcollections.LineCollection):
    def assemble(self, x, y, color='C0'):
            #assemble line marks as set of segments
            segments = [np.vstack((vx, vy)).T for vx, vy
                              in zip(x, y)]
            self.set_segments(segments)
            self.set_color(color)
```

To generate Fig. 11b, the `Line` artist `view` method returns a table of edges. Each edge consists of (x,y) points sampled along the line defined by the edge and information such as the color of the edge. As with `Point`, the data is then converted into visual variables. In `assemble`, this visual representation is composed into a set of line segments, where each segment is the array generated by `np.vstack((vx, vy))`. Then the colors of each line segment are set. The colors are guaranteed to correspond to the correct segment because of the atomicity constraint on view.

### 4.1.1 Visual Encoders

The visual parameter serves as the dictionary key because the visual representation is constructed from the encoding applied to the data $\mu = \nu \circ \tau$. For the scatter plot, the mappings for the visual fiber components $P = (x, y, facecolors, s)$ are defined as

```python
cmap =  color.Categorical({'true':'deeppink',
                           'false':'deepskyblue'})
transforms = {'x': {'name': 'v4', 'encoder': lambda x: x},
              'y': {'name': 'v2', 'encoder': lambda x: x},
              'facecolors': {'name':'v3', 'encoder': cmap},
              's':{'name': None ,
                   'encoder': lambda _: itertools.repeat(.02)}}
```

where `lambda x: x` is an identity $\nu$ `{'name':None}` maps into $P$ without corresponding $\tau$ to set a constant visual value, and `color.Categorical` is a custom $\nu$ implemented as a class for reusability. A test for equivariance can be implemented trivially

```python
def test_nominal(values, encoder):
    m1 = list(zip(values, encoder(values)))
    random.shuffle(values)
    m2 = list(zip(values, encoder(values)))
    assert sorted(m1) == sorted(m2)
```

but is currently factored out of the artist for clarity.

### 4.1.2 Data Model

The data input into the `Artist` will often be a wrapper class around an existing data structure. This wrapper object must specify the fiber components $F$ and connectivity $K$ and have a `view` method that returns an atomic object that encapsulates $\tau$. To support specifying the fiber bundle, we define a `FiberBundle` data class [2]

```python
@dataclass
class FiberBundle:
    K: dict #{'tables': []}
    F: dict #  {variable name: type}
```

that asks the user to specify the the properties of $F$ and the $K$ connectivity in terms of a simplicial triangulation scheme [31]. To generate the scatter plot and the line plot, the distinction is in the `tau` method that is the section.

```python
class VertexSimplex:
    """Fiberbundle is consistent across all sections
    """
    FB = FiberBundle({'tables': ['vertex']},
            {'v1': float, 'v2': str, 'v3': float})

    def __init__(self, sid = 45, size=1000, max_key=10**10):
        # create random list of keys
    def tau(self, k):
      # e1 is sampled from F1, e2 from F2, etc...
      return (k, (e1, e2, e3, e4))
```

The discrete `tau` method returns a record of discrete points, while the line`tau` returns a sampling of points along an edge $k$

```python
def tau(self, k): #will fix location on page on revision
    x, y = self._xy(k, self.distances,
               self.angle_samples[k], self.angle_samples[k+1])
    color = self._color(k)
    return (k, (x, y, color))
```

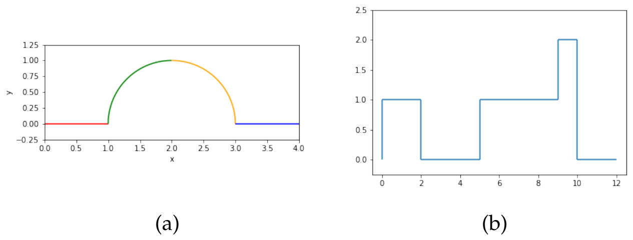
In both cases the `view` method packages the data

```python
def view(self, axes):
    table = defaultdict(list)
    for k in self.keys:
        table['index'].append(k)
        for (name, value) in zip(self.FB.fiber.keys(),
                                 self.tau(k)[1]):
            table[name].append(value)
    return table
```

into a data structure that the artist can unpack via data component name.

The graphics in figure 12 are made using the `Line` artist and the `Graphline` data source where if told that the data is connected, the data source will check for that connectivity by constructing an adjacency matrix. The multicolored line is a connected graph of edges with each edge function evaluated on 1000 samples,

```python
simplex.GraphLine(FB, edges, verticies,
                      num_samples=1000, connect=True)
```



(a) (b)

Fig. 12: Continuous and discontinuous lines as defined via the same data model, and generated with the same $A'$`Line`

while the stair chart is discontinuous and only needs to be evaluated at the edges of the interval

```python
simplex.GraphLine(FB, edges, verticies,
                      num_samples=2, connect=False)
```

such that one advantage of this model is it helps differentiate graphics that have different artists from graphics that have the same artist but make different assumptions about the source data.

## 5 Discussion

This work contributes a mathematical description of the mapping $A$ from data to visual representation. Combining Butler's proposal of a fiber bundle model of visualization data with Spivak's formalism of schema lets this model support a variety of datasets, including discrete relational tables,, multivariate high resolution spatio temporal datasets, and complex networks. Decomposing the artist into encoding $\nu$, assembly $Q$, and reindexing $\xi$, provides the specifications that the graphic must have continuity equivalent to the data, and that the visual characteristics of the graphics are equivariant to their corresponding components under monoid actions. This model defines these constraints on the transformation function such that they are not specific to any one type of encoding or visual characteristic. Encoding the graphic space as a fiber bundle provides a structure rich abstraction of the target graphical design in the target display space. The toy prototype built using this model validates that is usable for a general purpose visualization tool since it can be iteratively integrated into the existing architecture rather than starting from scratch. Factoring out graphic formation into assembly functions allows for much more clarity in how they differ. This prototype demonstrates that this framework can generate the fundemental point (scatter plot) and line (line chart) marks.

### 5.1 Limitations

So far this model has only been worked out for a single data set tied to a primitive mark. The examples and prototype have so far only been implemented for the static 2D case, but nothing in the math limits to 2D and expansion to the animated case should be possible because the model is formalized in terms of the sheaf. While this model supports equivariance of figurative glyphs generated from parameters of the data [12,23], it does not have a way to evaluate the semantic accuracy of the figurative representation. Even though the model is designed to be backend and format independent, it has only really been tested against PNGs rendered with the AGG backend. It is especially unknown how this framework interfaces with high performance rendering libraries such as openGL [25]. This model and the associated prototype is deeply tied to Matplotlib's existing architecture, so it has not been worked through how the model generalizes to other libraries, such as those built on Mackinlay's APT framework.

## 5.2 Future Work

More work is needed to formalize the composition operators and equivalence class $A'$. More artists need to be implemented to demonstrate that the model can underpin a minimally viable library, foremost an image [35, 36, 63], a heatmap [44, 68], and an imherently computational artist such as the boxplot [66]. This could be pushed further to integrate with topological [39] and functional [52] data analysis methods. Since this model formalizes notions of structure preservation, it can serve as a good base for tools that assess quality metrics [15] or invariance [42] of visualizations with respect to graphical encoding choices. While this paper formulates visualization in terms of monoidal action homomorphisms between fiberbundles, the model lends itself to a categorical formulation [29, 47] that could be further explored.

## 6 Conclusion

An unoffical philosophy of Matplotlib is to support making whatever kinds of plots a user may want, even if they seem nonsensical to the development team. The topological framework described in this work provides a way to facilitate this graph creation in a rigorous manner; any artist that meets the equivariance criteria described in this work by definition generates a graphic representation that matches the structure of the data being represented. We leave it to domain specialists to define the structure they need to preserve and the maps they want to make, and hopefully make the process easier by untangling these components into seperate constrained maps and providing a fairly general data and display model.

**Acknowledgments**

The authors wish to thank the Matplotlib development team for their invaluable feedback along the way, particulary Bruno Beltran for discussions about the visual fiber space. This work was supported in part by a grant from the Chan Zuckerberg Institute (EOSS cycle 1).

## References

[1] Action in nLab. https://ncatlab.org/nlab/show/action#actions_of_a_monoid.
[2] Dataclasses — Data Classes — Python 3.9.2rc1 documentation. https://docs.python.org/3/library/dataclasses.html.
[3] Locally trivial fibre bundle - Encyclopedia of Mathematics. https://encyclopediaofmath.org/wiki/Locally_trivial_fibre_bundle.
[4] Connected space. *Wikipedia*, Dec. 2020.
[5] Jet bundle. *Wikipedia*, Dec. 2020.
[6] Quotient space (topology). *Wikipedia*, Nov. 2020.
[7] Retraction (topology). *Wikipedia*, July 2020.
[8] Monoid. *Wikipedia*, Jan. 2021.
[9] Semigroup action. *Wikipedia*, Jan. 2021.
[10] Y. Albo, J. Lanir, P. Bak, and S. Rafaeli. Off the Radar: Comparative Evaluation of Radial Visualization Solutions for Composite Indicators. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):569–578, Jan. 2016. doi: 10.1109/TVCG.2015.2467322
[11] P. D. Auroux. Math 131: Introduction to Topology. p. 113.
[12] F. Beck. Software Feathers figurative visualization of software metrics. In *2014 International Conference on Information Visualization Theory and Applications (IVAPP)*, pp. 5–16, Jan. 2014.
[13] R. A. Becker and W. S. Cleveland. Brushing Scatterplots. *Technometrics*, 29(2):127–142, May 1987. doi: 10.1080/00401706.1987.10488204
[14] J. Bertin. *Semiology of Graphics : Diagrams, Networks, Maps*. ESRI Press, Redlands, Calif., 2011.
[15] E. Bertini, A. Tatu, and D. Keim. Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2203–2212, 2011.
[16] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, Nov. 2009. doi: 10.1109/TVCG.2009.174
[17] M. Bostock, V. Ogievetsky, and J. Heer. $D^3$ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185
[18] R. Brüggemann and G. P. Patil. *Ranking and Prioritization for Multi-Indicator Systems: Introduction to Partial Order Applications*. Springer Science & Business Media, July 2011.
[19] A. Buja, J. A. McDonald, J. Michalak, and W. Stuetzle. Interactive data visualization using focusing and linking. In *Proceedings of the 2nd Conference on Visualization '91*, VIS '91, pp. 156–163. IEEE Computer Society Press, Washington, DC, USA, 1991.
[20] D. M. Butler and S. Bryson. Vector-Bundle Classes form Powerful Tool for Scientific Visualization. *Computers in Physics*, 6(6):576, 1992. doi: 10.1063/1.4823118
[21] D. M. Butler and M. H. Pendley. A visualization model based on the mathematics of fiber bundles. *Computers in Physics*, 3(5):45, 1989. doi: 10.1063/1.168345
[22] L. Byrne, D. Angus, and J. Wiles. Acquired Codes of Meaning in Data Visualization and Infographics: Beyond Perceptual Primitives. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):509–518, Jan. 2016. doi: 10.1109/TVCG.2015.2467321
[23] L. Byrne, D. Angus, and J. Wiles. Figurative frames: A critical vocabulary for images in information visualization. *Information Visualization*, 18(1):45–67, Aug. 2017. doi: 10.1177/1473871617724212
[24] S. Carpendale. Visual Representation from Semiology of Graphics by J. Bertin.
[25] G. S. Carson. Standards pipeline: The OpenGL specification. *SIGGRAPH Comput. Graph.*, 31(2):17–18, May 1997. doi: 10.1145/271283.271292
[26] A. M. Cegarra. Cohomology of monoids with operators. In *Semigroup Forum*, vol. 99, pp. 67–105. Springer, 2019.
[27] C.-S. J. Chu. Time series segmentation: A sliding window approach. *Information Sciences*, 85(1):147–173, July 1995. doi: 10.1016/0020-0255(95)00021-G
[28] M. S. Crouch, A. McGregor, and D. Stubbs. Dynamic graphs in the sliding-window model. In *European Symposium on Algorithms*, pp. 337–348. Springer, 2013.
[29] B. Fong and D. I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, first ed., July 2019. doi: 10.1017/9781108668804
[30] M. Friendly. *A Brief History of Data Visualization*, pp. 15–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-33037-0_2
[31] J.-D. B. Geometrica. Simplicial Complexes. p. 52.
[32] B. Geveci, W. Schroeder, A. Brown, and G. Wilson. VTK. *The Architecture of Open Source Applications*, 1:387–402, 2012.
[33] R. Ghrist. Homological algebra and data. *Math. Data*, 25:273, 2018.
[34] R. W. Ghrist. *Elementary Applied Topology*, vol. 1. Createspace Seattle, 2014.
[35] R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in scientific computing*, 74:93, 1990.
[36] C. D. Hansen and C. R. Johnson. *Visualization Handbook*. Elsevier, 2011.
[37] M. D. Hanwell, K. M. Martin, A. Chaudhary, and L. S. Avila. The Visualization Toolkit (VTK): Rewriting the rendering code for modern graphics cards. *SoftwareX*, 1-2:9–12, Sept. 2015. doi: 10.1016/j.softx.2015.04.001
[38] J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, 2006. doi: 10.1109/TVCG.2006.178
[39] C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. A Survey of Topology-based Methods in Visualization. *Computer Graphics Forum*, 35(3):643–667, June 2016. doi: 10.1111/cgf.12933
[40] J. Hunter and M. Droettboom. The Architecture of Open Source Applications (Volume 2): Matplotlib. https://www.aosabook.org/en/matplotlib.html.
[41] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science Engineering*, 9(3):90–95, May 2007. doi: 10.1109/MCSE.2007.55
[42] G. Kindlmann and C. Scheidegger. An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2181–2190, Dec. 2014. doi: 10.1109/TVCG.2014.

2346325

[43] W. A. Lea. A formalization of measurement scale forms. p. 44.

[44] T. Loua. *Atlas Statistique de La Population de Paris*. J. Dejey & cie, 1873.

[45] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, Apr. 1986. doi: 10.1145/22949.22950

[46] J. Mackinlay. *Automatic Design of Graphical Presentations*. PhD Thesis, Stanford, 1987.

[47] B. Milewski. Category Theory for Programmers. p. 498.

[48] T. Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC press, Oct. 2014.

[49] J. Musilová and S. Hronek. The calculus of variations on jet bundles as a universal approach for a variational formulation of fundamental physical theories. *Communications in Mathematics*, 24(2):173–193, Dec. 2016. doi: 10.1515/cm-2016-0012

[50] D. Nekrasovski, A. Bodnar, J. McGrenere, F. Guimbretière, and T. Munzner. An evaluation of pan &amp; zoom and rubber sheet navigation with and without an overview. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pp. 11–20. Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1124772.1124775

[51] Z. Qu and J. Hullman. Keeping multiple views consistent: Constraints, validations, and exceptions in visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):468–477, Jan. 2018. doi: 10.1109/TVCG.2017.2744198

[52] J. O. Ramsay. *Functional Data Analysis*. Wiley Online Library, 2006.

[53] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, pp. 669–678. ACM, Honolulu Hawaii USA, Oct. 2014. doi: 10.1145/2642918.2647360

[54] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675, July 2012. doi: 10.1038/nmeth.2089

[55] N. Sofroniew, T. Lambert, K. Evans, P. Winston, J. Nunez-Iglesias, G. Bokota, K. Yamauchi, A. C. Solak, ziyangczi, G. Buckley, M. Bussonnier, D. D. Pop, T. Tung, V. Hilsenstein, Hector, J. Freeman, P. Boone, alisterburt, A. R. Lowe, C. Gohlke, L. Royer, H. Har-Gil, M. Kittisopikul, S. Axelrod, kir0ul, A. Patil, A. McGovern, A. Rokem, Bryant, and G. Peña-Castellanos. Napari/napari: 0.4.5rc1. Zenodo, Feb. 2021. doi: 10.5281/zenodo.4533308

[56] E. Spanier. *Algebraic Topology*. McGraw-Hill Series in Higher Mathematics. Springer, 1989.

[57] D. I. Spivak. SIMPLICIAL DATABASES. p. 35.

[58] D. I. Spivak. Databases are categories, June 2010.

[59] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.

[60] M. Stieven. A Monad is just a Monoid. . . . https://medium.com/@michelestieven/a-monad-is-just-a-monoid-a02bd2524f66, Apr. 2020.

[61] S. Studies. Culturevis/imageplot, Jan. 2021.

[62] T. Sugibuchi, N. Spyratos, and E. Siminenko. A framework to analyze information visualization based on the functional data model. In *2009 13th International Conference Information Visualisation*, pp. 18–24, 2009. doi: 10.1109/IV.2009.56

[63] M. Tory and T. Moller. Rethinking visualization: A high-level taxonomy. In *IEEE Symposium on Information Visualization*, pp. 151–158, 2004. doi: 10.1109/INFVIS.2004.59

[64] J. VanderPlas, B. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software*, 3(32):1057, Dec. 2018. doi: 10.21105/joss.01057

[65] H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[66] H. Wickham and L. Stryjewski. 40 years of boxplots. *The American Statistician*, 2011.

[67] L. Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer-Verlag New York, Inc., New York, 2nd ed ed., 2005.

[68] L. Wilkinson and M. Friendly. The History of the Cluster Heat Map. *The American Statistician*, 63(2):179–184, May 2009. doi: 10.1198/tas.2009.0033

[69] K. Wongsuphasawat. Navigating the Wide World of Data Visualization Libraries (on the web), 2021.

[70] B. A. Yorgey. Monoids: Theme and Variations (Functional Pearl). p. 12.

[71] C. Ziemkiewicz and R. Kosara. Embedding Information Visualization within Visual Representation. In Z. W. Ras and W. Ribarsky, eds., *Advances in Information and Intelligent Systems*, pp. 307–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-04141-9_15