1 Topological Artist Model

To guide the implementation of structure preserving building block components, we develop a mathematical formalism of visualization that specifies how these components preserve *continuity* and *equivariance*. Inspired by the somewhat analogous component in Matplotlib[1], we call the transformation from data space to graphic that these building block components implement the *artist*.

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

The *artist* $\mathscr{A}$ is a map from the data $\mathscr{E}$ to graphic $\mathscr{H}$ fiber bundles. To explain how the *artist* is a structure preserving map from data to graphic, we first describe how we model data (subsection 1.1) and graphics (subsection 1.2) as topological structures that encapsulate component types and continuity. We then discuss the maps from graphic to data (subsubsection 1.2.2, data components to visual components (subsubsection 1.3.2), and visual components into graphic (subsubsection 1.3.3) that make up the artist.

1.1 Data Space E

Building on Butler's proposal of using fiber bundles as a common data representation structure for visualization data[2, 3], a fiber bundle is a tuple (E, K, π, F) defined by the projection map π

$$F \hookrightarrow E \xrightarrow{\pi} K \tag{2}$$

that binds the components of the data in F to the continuity of the data encoded in K. The fiber bundle models the properties of data component types F (subsubsection 1.1.1), the continuity of records K (subsubsection 1.1.3), the collections of records τ (**??**), and the space E of all possible datasets with these components and continuity. By definition fiber bundles are locally trivial[4, 5], meaning that over a localized neighborhood U the total space is the cartesian product $K \times F$. We use fiber bundles as the data model because they are inclusive enough to express all the types of structures of data described in **??**

1.1.1 Variables in Fiber Space F

To formalize the structure of the data components, we use notation introduced by Spivak [6] that binds the components of the fiber to variable names. This allows us to describe the components in a schema like way. 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 $\mathbb{U}_\sigma$

$$\begin{tikzcd} \mathbb{U}_\sigma \arrow[r] \arrow[d, "\pi_\sigma"'] & \mathbb{U} \arrow[d, "\pi"] \\ C \arrow[r, "\sigma"'] & \mathbf{DT} \end{tikzcd} \tag{3}$$

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 σ, the fiber for a one variable dataset is

$$F = \mathbb{U}_{\sigma(c)} = \mathbb{U}_T \tag{4}$$

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

$$F = \mathbb{U}_{\sigma(c_1)} \times \ldots \mathbb{U}_{\sigma(c_i)} \ldots \times \mathbb{U}_{\sigma(c_n)} \tag{5}$$

which is equivalent to

$$F = F_0 \times \ldots \times F_i \times \ldots \times F_n \tag{6}$$

which allows us to decouple F into components F_i.

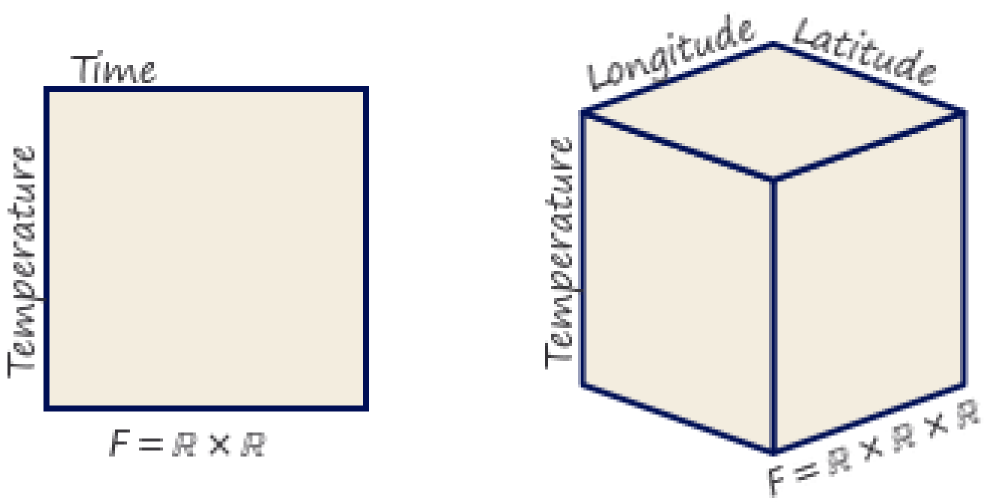


Figure 1: The fiber space is the set of all combinations of all theoretically possible values of the components. The 2D fiber $F = \mathbb{R} \times \mathbb{R}$ encodes the properties of *time* and *temperature* components. One dimension of the fiber encodes the range of possible values for the time component of the dataset, which is a subset of the $\mathbb{R}$, while the other dimension encodes the range of possible values $\mathbb{R}$ for the temperature component. This means the fiber is the set of points *(temperature, time)*that are all the combinations of *temperatature* $\times$ *time*. The 3D fiber encodes points at all possible combinations of *temperature*, *latitude*, and *longitude*.

For example, the records in the 2D fiber in **??** are a pair of *times* and °K *temperature* measurements taken at those times. Time is a positive number of type `datetime` which can be resolved to floats $\mathbb{U}_{\texttt{datetime}} = \mathbb{R}$. Temperature values are real positive numbers $\mathbb{U}_{\texttt{float}} = \mathbb{R}^+$. The fiber is

$$F = \mathbb{R} \times \mathbb{R}^+$$

where the first component F_0 is the set of values specified by ($c = time$, $T =$ `datetime`, $\mathbb{U}_\sigma = \mathbb{R}$) and F_1 is specified by ($c = temperature$, $T =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$) and is the set of values $\mathbb{U}_\sigma = \mathbb{R}$. In the 3D fiber in **??**, time is replaced with location. This location variable is of type `point` and has two components *latitude* and *longitude* $\{(lat, lon) \in \mathbb{R}^2 \mid -90 \leq lat \leq 90,\ 0 \leq lon \leq 360\}$. The fiber for this dataset is

$$F = \mathbb{R} \times \mathbb{R}^2 = \mathbb{R} \times \mathbb{R} \times \mathbb{R}$$

where the dimensionality of the fiber does not change, but the components of the fiber can be coupled. For example, *location* can can either be specified as ($c = location$, $T =$ `point`, $\mathbb{U}_\sigma = \mathbb{R}^2$)or ($c = latitude$, $T =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$) and ($c = longitude$, $T =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$).

As illustrated in Figure 1, Spivak's framework provides a consistent way to describe potentially complex components of the input data.

1.1.2 Measurement Scales: Monoid Actions

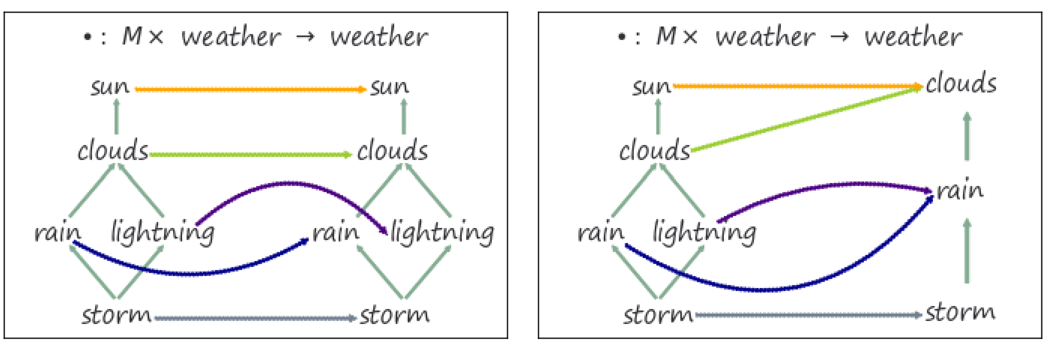
Implementing expressive visual encodings requires formally describing the structure on the components of the fiber, which we define by the actions of a monoid on the component. In doing so, we specify the properties of the component that must be preserved in a graphic representation. A monoid [7] M is a set with a binary operation $* : M \times M \rightarrow M$ that satisfies the axioms:

$$\textbf{associativity} \text{ for all} a, b, c \in M \; (a \bullet b) \bullet c = a \bullet (b \bullet c)$$
$$\textbf{identity} \text{ for all } a \in M, \; e \bullet a = a$$

As defined on a component of F, a left monoid action [8, 9] of M_i is a set F_i with an action $\bullet : M \times F_i \rightarrow F_i$ with the properties:

$$\textbf{associativity} \text{ for all } f, g \in M_i \text{ and } x \in F_i, \; f \bullet (g \bullet x) = (f * g) \bullet x$$
$$\textbf{identity} \text{ for all } x \in F_i, e \in M_i, \; e \bullet x = x$$

The identity and associativity properties of the action denote that the action is a monoid homomorphism, which means that the group operation is preserved on both sides of the action[10].



(a) (b)

Figure 2: The action $\bullet$ in **??** is the arrows from the partial order diagram of weather states on the left to the diagram of weather states on the right. Since the action maps the weather states to themselves, the ordering defined by the monoid $*$ is preserved on both sides of the action. The action in **??** is monoid homomorphism because the ordering of the weather states is the same as the ordering of the elements they are mapped to. Given $sun \geq clouds \geq rain\, lightining$ on the right, the action $sun\, clouds \rightarrow clouds$, and $rain\, lightining \rightarrow rain$ is structure preserving because on the left $cloud \geq rain$ so the relative ordering of elements is the same as the elements they are mapped to.

One example of monoids are partial orderings on a set, such as seen in . Each hasse diagram of the set of weather states describes an ordering on the set; the arrow goes from the lesser value to the greater one. For example, $storm \leq rain$. In **??**, the action $\bullet$ maps the elements of a set of weather states into itself by mapping them into other elements of the weather states. The action in Figure 2a, represented as the arrows between the hasse diagrams of the weather states, maps the weather states to themselves; therefore the ordering of the weather states is identical on both sides of the action and it is therefore homomorphic. The action $\bullet$ in Figure 2b is a monotone map[11]

$$if\ a \leq b\ then\ \bullet(a) \leq \bullet(b) \mid a, b \in F_i$$

where the structure the action preserves is the relative, rather than exact, ordering. Since groups are monoids with invertible operations, this definition of structure is also broad enough to include the Steven's measurment scales[12, 13]. Monoids are also commonly found in functional programming since the core property of monoids is composability [14].

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

$$M = M_0 \times \ldots \times M_i \times \ldots \times \ldots M_n \tag{7}$$

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

1.1.3 Continuity of the Data K

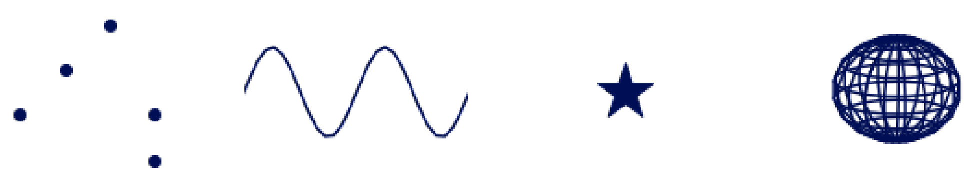


Figure 3: The topological base space K encodes the continuity of the data space, for example if the data is discrete points or lies on a plane or a sphere

The base space K acts as an indexing space, as emphasized by Butler[2, 3], to express how the records in E are connected to each other. As shown in Figure 3, K can have any number of dimensions and can be continuous or discrete. The base space also does not describe anything about the dataset besides the continuity. While the base space may have components to identify the continuity, such as *time, latitutde, longitude*, these labels are indexed into from K the same as any other component. This is similar to the notion of structural *keys* with associated *values* proposed by Munzner[15], but our model treats keys as a pure reference to topology. Decoupling the keys from their semantics allows the metadata to be altered; this provides for coordinate agnostic representation of the continuity and facilitates encoding of data where the independent variable may not be clear. For example the amount of snow on the ground is dependent on time of day and how much snow has fallen, and changing the coordinate system or time resolution should have no effect on how the records are connected to each other.

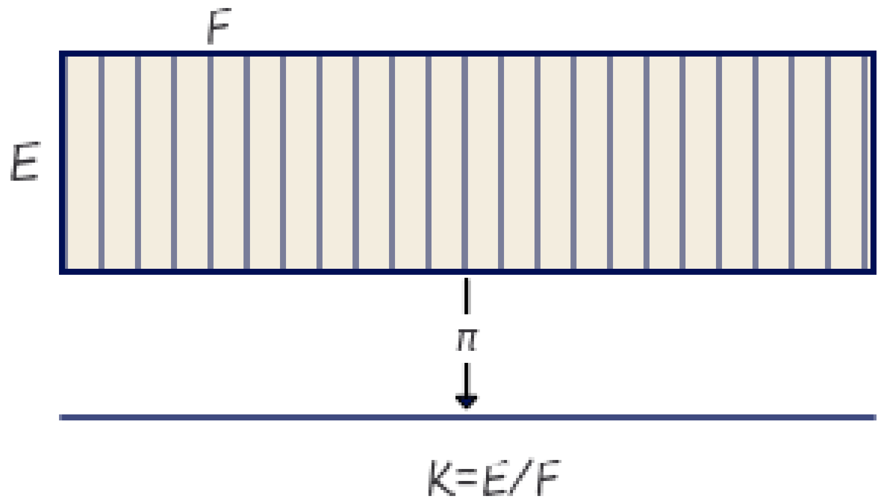


Figure 4: The base space E is divided into fiber segments F. The base space K acts as an index into the records in the fibers.

Formally K is the quotient space [16] of E meaning it is the finest space[17] such that every $k \in K$ has a corresponding fiber F_k[16]. In Figure 4, E is a rectangle divided by vertical fibers F, so the minimal K for which there is always a mapping $\pi : E \to K$ is the closed interval $[0, 1]$.

As with Equation 6 and Equation 7, we can decompose the total space into component bundles$\pi : E_i \to K$ where

$$\pi : E_1 \oplus \ldots \oplus E_i \oplus \ldots \oplus E_n \to K \tag{8}$$

such that M_i acts on component bundle E_i. The K remains the same because the connectivity of records does not change just because there are fewer components 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.

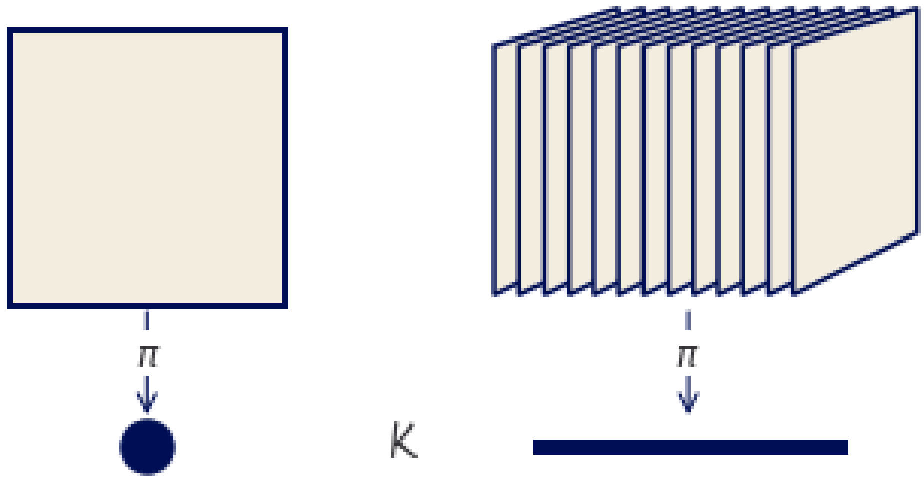


Figure 5: These two datasets have the same *(time, temperature)* fiber, but different continuities. The dataset on the left consists of discrete records, while the records in the dataset on the right sampled from a continuous space.

The datasets in Figure 5 have the same fiber of (temperature, time). The dot represents a discrete base space K, meaning that every dataset encoded in the fiber bundle has discrete continuity. The line is a representation of a 1D continuity, meaning that every dataset in the fiber bundle is 1D continous. 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.

1.1.4 Data τ

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 example, in the special case of a table [6], K is a set of row ids, F is the columns, and the section τ returns the record r at a given key in K. For any fiber bundle, there exists a map

$$\begin{tikzcd} F \arrow[r, hook] & E \arrow[d, "\pi"'] \\ & K \arrow[u, bend right, "\tau"'] \end{tikzcd} \tag{9}$$

such that $\pi(\tau(k)) = k$. The set of all global sections is denoted as $\Gamma(E)$. Assuming a trivial fiber bundle $E = K \times F$, the section is

$$\tau(k) = (k, (g_{F_0}(k), \ldots, g_{F_n}(k))) \tag{10}$$

where $g : K \to F$ is the index function into the fiber. This formulation of the section also holds on locally trivial sections of a non-trivial fiber bundle. Because we can decompose the bundle and the fiber (Equation 8, Equation 6), we can decompose τ as

$$\tau = (\tau_0, \ldots, \tau_i, \ldots, \tau_n) \tag{11}$$

where each section τ_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.

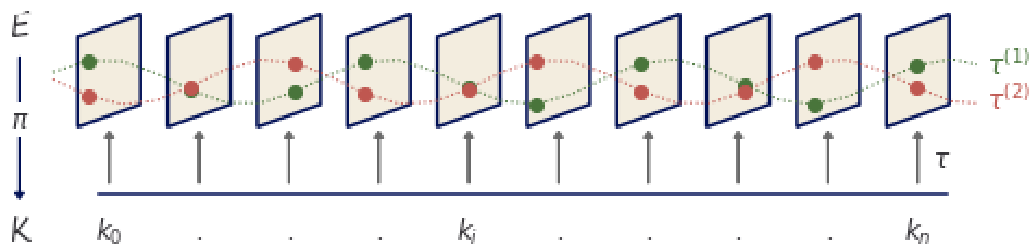


Figure 6: Fiber (time, temperature) with an interval K basespace. The sections $\tau^{(1)}$ and $\tau^{(2)}$ are constrained such that the time variable must be monotonic, which means each section is a timeseries of temperature values. They are included in the global set of sections $\tau^{(1)}, \tau^{(2)} \in \Gamma(E)$

In Figure 6, the fiber is the same encoding of ($time$, $temperature$) illustrated in Figure 1, and the base space is the interval K shown in Figure 5. The section $\tau^{(1)}$ is a function that for a point k returns a record in the fiber F. The section applied to a set of points in K resolves to a series of monotonically increasing in time records of *(time, temperature)* values. Section $\tau^{(2)}$ returns a different timeseries of *(time, temperature)* values. Both sections are included in the global set of sections $\tau^{(1)}, \tau^{(2)} \in \Gamma(E)$.

1.1.5 Sheafs

Many types of dynamic visualizations require evaluating sections on different subspaces of K, an a sheaf, denoted $\mathcal{O}$ provides a way to do so. A sheaf is a mathematical structure for defining collections of objects[18–20] on mathematical spaces. On the fiber bundle E, we can describe a sheaf as the collection of local sections $\iota^*\tau$

$$\begin{tikzcd} \iota^* E \arrow[r, hook, "\iota^*"] \arrow[d, shift right, "\pi"'] & E \arrow[d, shift right, "\pi"'] \\ U \arrow[r, hook, "\iota"] \arrow[u, shift right, bend right, "\iota^*\tau"'] & K \arrow[u, shift right, bend right, "\tau"'] \end{tikzcd} \tag{12}$$

which are sections of E pulled back over local neighborhood $U \subset E$ via the inclusion map $\iota : E \to U$. The collation of sections enabled by sheafs is necessary for navigation techniques such as pan and zoom[21] and dynamically updated visualizations such as sliding windows[22, 23].

1.1.6 Applications to Data Containers

This model provides a common formalism for widely used data containers without sacrificing the semantic structure embedded in each container. For example, the section can be any instance of a univariate numpy array[24] that stores an image. This could be a section of a fiber bundle where K is a 2D continuous plane and the F is $(\mathbb{R}^3, \mathbb{R}, \mathbb{R})$ where $\mathbb{R}^3$ is color, and the other two components are the x and y positions of the sampled data in the image. This position information is already implicitely encoded in the array as the index and the resolution of the image being stored.Instead of an image, the numpy array could also store a 2D discrete table. The fiber would not change, but the K would now be 0D discrete points. These different choices in topology indicate, for example, what sorts of interpolation would be appropriate when visualizing the data.

There are also many types of labeled containers that can richly be described in this framework because of the schema like structure of the fiber. For example, a pandas series which stores a labeled list, or a dataframe[25] which stores a relational table. A series could store the values of $\tau^{(1)}$ and a second series could be $\tau^{(2)}$. We could also fatten the fiber to hold two temperature series, such that a section would be an instance of a dataframe with a time column and two temperature columns. While the series and dataframe explicitly have a time index column, they are components in our model and the index is assumed to be data independent references such as hashvalues, virtual memory locations, or random number keys.

Where this model particularly shines are N dimensional labeled data structures. For example, an xarray[26] data that stores temperature field could have a K that is a continuous volume and the components would be the temperature and the time, latitude, and longitude the measurements were sampled at. A section can also be an instance of a distributed data container, such as a dask array [27]. As with the other containers, K and F are defined in terms of the index and dtypes of the components of the array. Because our framework is defined in terms of the fiber, continuity, and sections, rather than the exact values of the data, our model does not need to know what the exact values are until the renderer needs to fill in the image.

1.2 Graphic Space H

To establish that the artist is structure preserving map from data E to graphic H we construct a graphic bundle so that we can define *equivariance* in terms of maps on the fiber spaces and *continuity* in terms of maps on the base space. As with the data, we can represent the target graphic as a section ρ of a bundle (H, S, π, D).

$$\begin{tikzcd} D \arrow[r, hook] & H \arrow[d, "\pi"'] \\ & S \arrow[u, bend right, "\rho"'] \end{tikzcd} \tag{13}$$

The graphic bundle H consists of a base S(subsubsection 1.2.1) that is a thickened form of K a fiber D(subsubsection 1.2.2) that is an idealized display space, and sections ρ(**??**) that encode a graphic where the visual characteristics are fully specified.

1.2.1 Idealized Display D

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. To support overplotting and transparency, the fiber could be $D = \mathbb{R}^7$ such that $(x, y, z, r, g, b, a) \in D$ specifies the target display. By abstracting the target display space as D, the model can support different targets, such as a 2D screen or 3D printer.

1.2.2 Continuity of the Graphic S

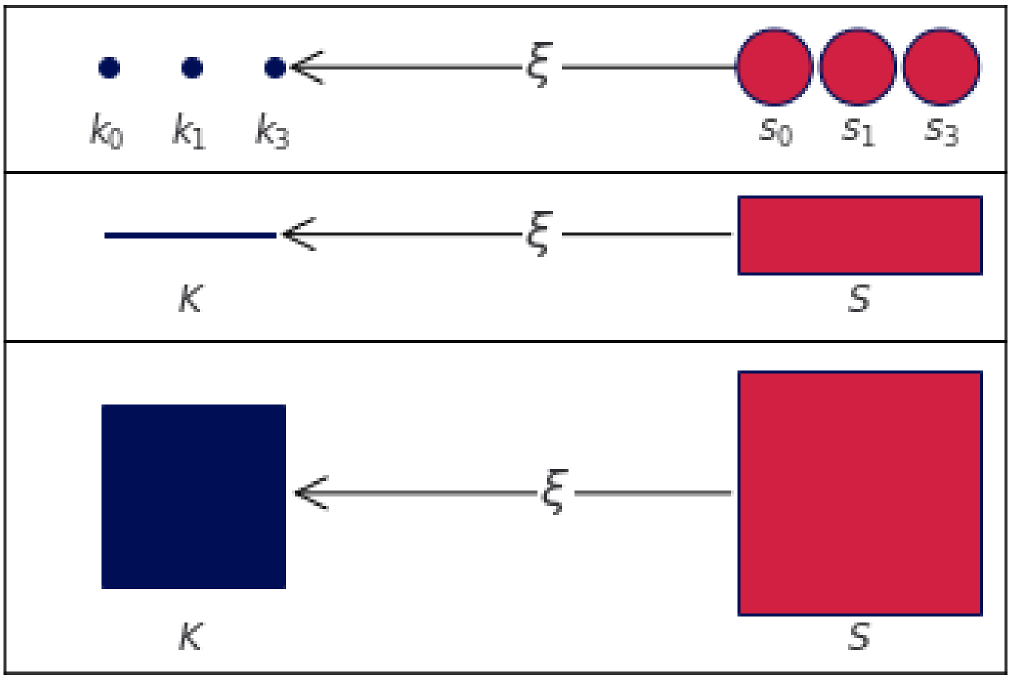


Figure 7: For a visualization component to preserve continuity, it must have a continuous surjective map $\xi : S \to K$ from graphic continuity to data continuity. The scatter and line graphic base spaces S have one more dimension of continuity than K so that S can encode physical aspects of the glyph, such as shape (a circle) or thickness. The image has the same dimension in S as in K.

To establish that a visualization component preserves continuity, we propose that their must be a continuous map $\xi : S \to K$ from the graphic base space to the data space. For example, consider a S that is mapped to the region of a 2D display space that represents K. For some visualizations, K may be lower dimension than S. For example, a point that is 0D in K cannot be represented on screen unless it is thickened to 2D to encode the connectivity of the pixels that visually represent the point. This thickening is often not necessary when the dimensionality of K matches the dimensionality of the target space, for example if K is 2D and the display is a 2D screen. We introduce S to thicken K in a way which preserves the structure of K.

Formally, we require that K be a deformation retract[28] of S so that K and S have the same homotopy, meaning there is a continuous map from S to K[29]. The surjective map

$\xi : S \to K$

$$\begin{tikzcd} E \arrow[d, "\pi"'] & H \arrow[d, "\pi"'] \\ K & S \arrow[l, "\xi"'] \end{tikzcd} \tag{14}$$

goes from region $s \in S_k$ to its associated point s. This means that if $\xi(s) = k$, the record at k is copied over the region s such that $\tau(k) = \xi^*\tau(s)$ where $\xi^*\tau(s)$ is τ pulled back over S. When K is discrete points and the graphic is a scatter plot, each point $k \in K$ corresponds to a 2D disk S_k as shown in Figure 7. In the case of 1D continuous data and a line plot, the region β over a point α_i specifies the thickness of the line in S for the corresponding τ on k. The image has the same dimensions in data space and graphic space such that no extra dimensions are needed in S.

The mapping function ξ provides a way to identify the part of the visual transformation that is specific to the the connectivity of the data rather than the values; for example it is common to flip a matrix when displaying an image. The ξ mapping is also used by interactive visualization components to look up the data associated with a region on screen. One example is to fill in details in a hover tooltip, another is to convert region selection (such as zooming) on S to a query on the data to access the corresponding record components on K.

1.2.3 Graphic ρ

The section $\rho : S \to H$ is the graphic in an idealizes prerender space and also acts as a specification for rendering the graphic to an image. It is sufficient to sketch out how an arbitrary pixel would be rendered, where a pixel p in a real display corresponds to a region S_p in the idealized display. To determine the color of the pixel, we aggregate the color values over the region via integration:

$$r_p = \iint\limits_{S_p} \rho_r(s) ds^2$$
$$g_p = \iint\limits_{S_p} \rho_g(s) ds^2$$
$$b_p = \iint\limits_{S_p} \rho_b(s) ds^2$$

For a 2D screen, the pixel is defined as a region $p = [y_{top}, y_{bottom}, x_{right}, x_{left}]$ of the rendered graphic. Since the x and y in p are in the same coordinate system as the x and y components of D the inverse map of the bounding box $S_p = {\rho_{xy}}^{-1}(p)$ is a region $S_p \subset S$. The color is the result of the integration over S_p.

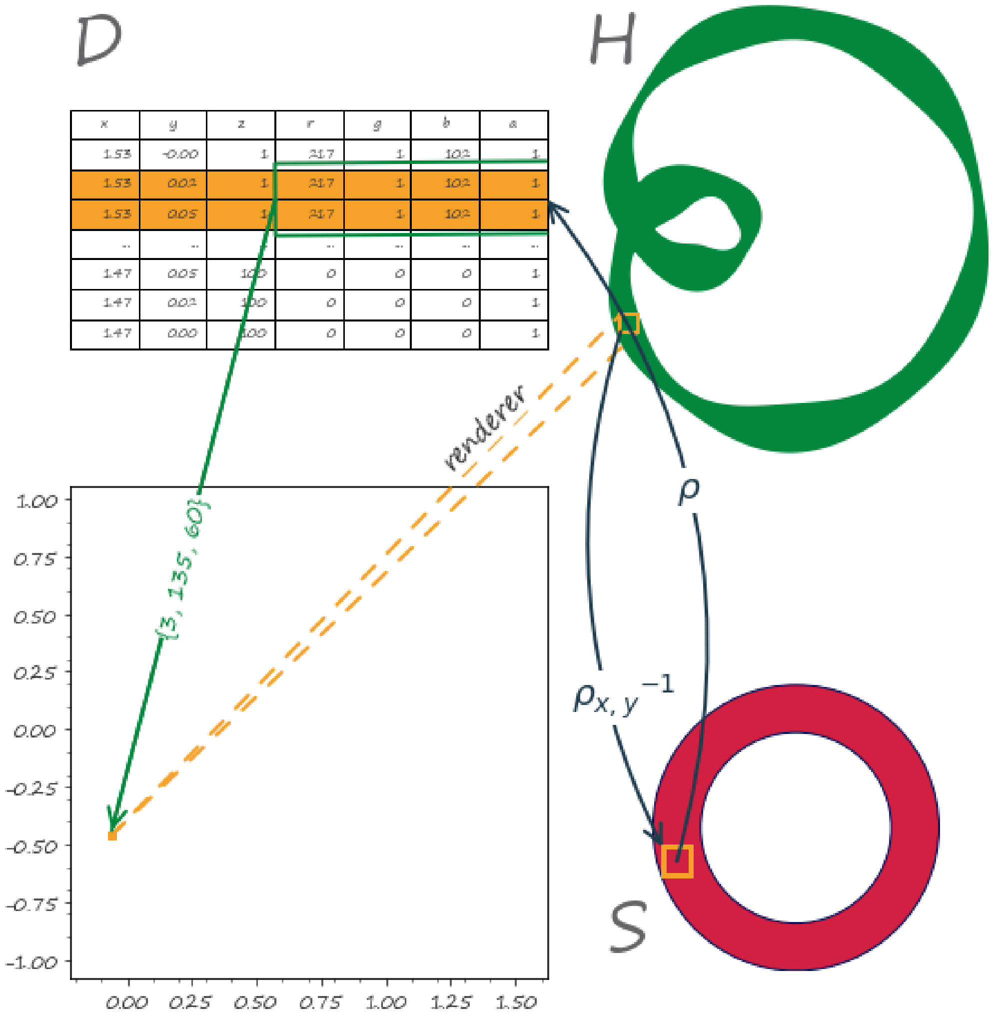


Figure 8: To render a graphic, a pixel p is selected in the display space, which is defined in the same coordinates as the x and y components in D via the renderer. In H the inverse mapping $\rho_{xy}{}^{(}p)$ returns a region $S_p \subset S$. $\rho(S_p)$ returns a set of points $(x, y, r, g, b) \in D$ that lie over S_p. The integral over the (r, g, b) pixels specifies that the pixel should be green

As shown in Figure 8, a pixel p in the output space, drawn in yellow, is selected and mapped, via the renderer, into a region on H. The region on H corresponds to a region $S_p \subset S$ via the inverse mapping $\rho_{xy}{}^{(}p)$. The base space S is an annulus to match the topology of the graphic idealized in H. The section $\rho(S_p)$ then maps into the fiber D over S_p to obtain the set of points in D, here represented as a table, that correspond to that section. The integral over the pixel components of this set of points in the fiber yields $\{3,\ 135,\ 60\}$ the actual color of the pixel. In general, ρ is an abstraction of rendering. In very broad strokes ρ can be a specification such as PDF[30], SVG[31], or an openGL scene graph[32]. Alternatively, ρ can be a rendering engine such as cairo[33] or AGG[34]. Implementation of ρ is out of scope for this work,

1.3 Artist

The topological artist A is how we model the building block component that transforms data into a graphic. The artist A is a map from the sheaf on a data bundle E which is $\mathcal{O}(E)$ to the sheaf on the graphic bundle H, $\mathcal{O}(H)$.

$$A : \mathcal{O}(E) \to \mathcal{O}(H) \tag{15}$$

The artist preserves *continuity* through the ξ map discussed in subsubsection 1.2.2 and is an *equivariant* map because it carries a homomorphism of monoid actions [35]

$$\varphi : M \to M' \tag{16}$$

Given M on data $\mathscr{E}$ and M' on graphic $\mathscr{H}$, we propose that artists $\mathscr{A}$ are equivariant maps

$$A(m \cdot r) = \varphi(m) \cdot A(r) \tag{17}$$

such that applying a monoid action $m \in M$ to the data input $r \in \mathscr{E}$ of the artist $\mathscr{A}$ is equivalent to applying a monoid action $\varphi(M) \in M'$ to the graphic $A(r) \in \mathscr{H}$ output of the artist.

The monoid equivariant map has two stages: the encoders $\nu : E' \to V$ convert the data components to visual components, and the assembly function $Q : \xi^*V \to H$ composites the fiber components of ξ^*V into a graphic in H.

$$\begin{tikzcd} E' \arrow[r, "\nu"] \arrow[dr, "\pi"'] & V \arrow[d, "\pi"] & \xi^*V \arrow[l, "\xi^*"'] \arrow[r, "Q"] \arrow[d, "\xi^*\pi"'] & H \arrow[dl, "\pi"] \\ & K & S \arrow[l, "\xi"] & \end{tikzcd} \tag{18}$$

ξ^*V is the visual bundle V pulled back over S via the equivariant continuity map $\xi : S \to K$ introduced in subsubsection 1.2.2. The functional decomposition of the visualization artist in Equation 18facilitates building reusable components at each stage of the transformation because the equivariance constraints are defined on ν, Q, and ξ. We name this map the artist as that is the analogous part of the Matplotlib[1] architecture that builds visual elements.

1.3.1 Visual Fiber Bundle V

We introduce a visual bundle V to store the mappings of the data components into components of the graphic. The visual bundle (V, K, π, P) is the 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 π

$$\begin{tikzcd} P \arrow[r, hook] & V \arrow[d, "\pi"'] \\ & K \arrow[u, bend right, "\mu"'] \end{tikzcd} \tag{19}$$

where μ is the visual variable encoding, as described by Bertin [36], of the data section τ. 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.

$\boldsymbol{\nu_i}$	$\boldsymbol{\mu_i}$	$\boldsymbol{codomain(\nu_i) \subset P_i}$
position	x, y, z, theta, r	$\mathbb{R}$
size	linewidth, markersize	$\mathbb{R}^+$
shape	markerstyle	$\{f_0, \ldots, f_n\}$
color	color, facecolor, markerfacecolor, edgecolor	$\mathbb{R}^4$
texture	hatch	$\mathbb{N}^{10}$
	linestyle	$(\mathbb{R}, \mathbb{R}^{+n, n\%2=0})$

Table 1: Some possible components of the fiber P for a visualization function implemented in Matplotlib

A section μ is a tuple of visual values that specifies the visual characteristics of a part of the graphic. For example, given a fiber of $\{x, y, color\}$ one possible section could be $\{.5, .5, (255, 20, 147)\}$. The $codomain(\nu_i)$ determines which monoids can act on P_i. These fiber components are implicit in the library, as seen in Table 1, and by making them explicit as components of the fiber we can build consistent definitions and expectations of how these parameters behave.

1.3.2 Visual Encoders ν

We define the visual transformers ν

$$\{\nu_0, \ldots, \nu_n\} : \{\tau_0, \ldots, \tau_n\} \mapsto \{\mu_0, \ldots, \mu_n\} \tag{20}$$

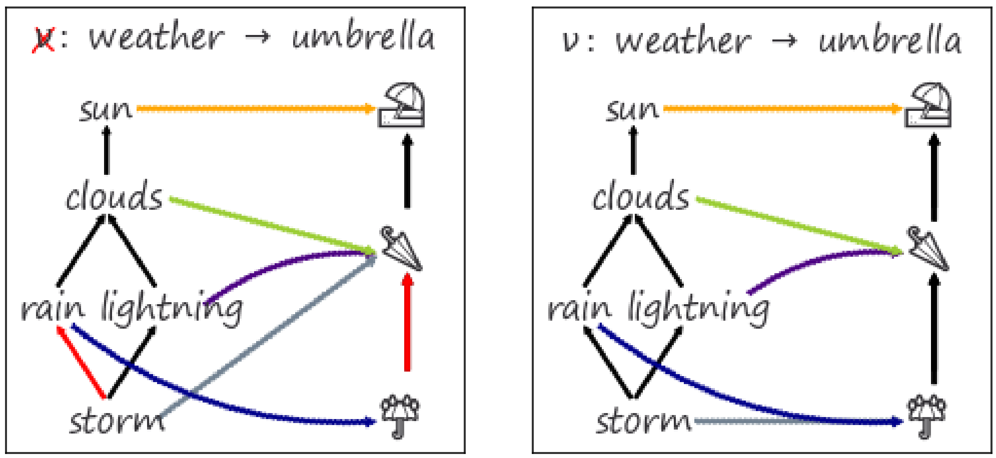
as the set of equivariant maps $\nu_i : \tau_i \mapsto \mu_i$. Given M_i is the monoid action on E_i and that there is a monoid $M_i{}'$ on V_i, then there is a monoid homomorphism from $\varphi : M_i \rightarrow M_i{}'$ that ν must preserve. As mentioned in subsubsection 1.1.2, monoid actions define the structure on the fiber components and are therefore the basis for equivariance. A validly constructed ν is one where the diagram of the monoid transform m commutes

$$\begin{tikzcd} E_i \arrow[r, "\nu_i"] \arrow[d, "m_r"'] & V_i \arrow[d, "m_v"] \\ E_i \arrow[r, "\nu_i"] & V_i \end{tikzcd} \tag{21}$$

such that applying equivariant monoid actions to E_i and V_i preserves the map $\nu_i : E_i \rightarrow V_i$. 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 ν_i must satisfy the constraint

$$\nu_i(m_r(E_i)) = \varphi(m_r)(\nu_i(E_i)) \tag{22}$$

such that φ 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 φ that takes M to M'. This allows us to define a monoid action on P of M that is $(m, v) \to \varphi(m) \bullet v$. 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 φ is the identity function.



(a) (b)

Figure 9: The map from data component to visual component in Figure 9a is not homomorphic, and therefore invalid, because $rain \geq storm$ is mapped to elements with the reverse ordering $\nu(storm) \geq \nu(storm)$. In contrast, the mapping in Figure 9b is valud since $\nu(storm) = \nu(rain)$ satisfies the condition $\nu(storm) \geq \nu(storm)$

The translation from weather state data to visual representation as umbrella emoji in Figure 9a is an invalid visual encoding map ν because it is not homomorphic. This is because the monotonic condition $rain \geq storm \implies \nu(rain) \geq \nu(storm)$ is not met since $\nu(rain) \leq \nu(storm)$. To satisfy the monotonic condition for $rain \geq storm$, either red arrow in Figure 9a would have to go in a different direction. On the other hand, the mapping from weather state to umbrellla in Figure 9b is a homomorphism since $\nu(rain) = \nu(storm)$ satisfies the monotonic condition of $rain \geq storm$. Figure 9 is an example of how the model supports partially ordered data components, which was a motivation for defining equivariance as monoid homomorphisms.

scale	**group**	**constraint**
nominal	permutation	if $r_1 \neq r_2$ then $\nu(r_1) \neq \nu(r_2)$
ordinal	monotonic	if $r_1 \leq r_2$ then $\nu(r_1) \leq \nu(r_2)$
interval	translation	$\nu(x + c) = \nu(x) + c$
ratio	scaling	$\nu(xc) = \nu(x) * c$

Table 2

The Stevens measurement types[12], listed in Table 2, are specified in terms of groups, which are monoids with invertible operations[37]. Despite critiques of the scales[38, 39], we believe it is critical for the model to include the measurement scales since they are commonly used in visualization to classify components [15, 40]. By specifying the equivariance constraints on ν we can guarantee that the stage of the artist that transforms data components into visual representations is equivariant. These constraints guide the implementation of reusable component transformers ν that are composed when generating the graphic.

1.3.3 Visualization Assembly

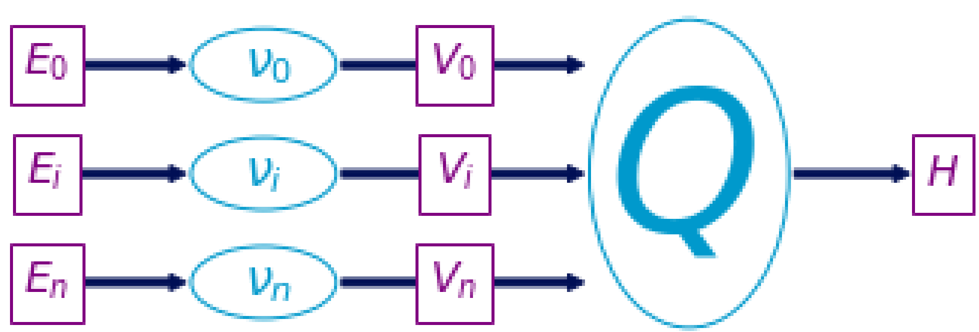


Figure 10: The transform functions ν_i convert data $\tau_i \in E$ to visual characteristics $\mu_i \in V$, then Q assembles μ_i into a graphic $\rho \in H$.

The transformation from data into graphic is analagous to a map-reduce operation; as illustrated in **??**, data components E_i are mapped into visual components V_i that are reduced into a graphic in H. The space of all graphics that Q can generate is the subset of graphics reachable via applying the reduction function $Q(\Gamma(V)) \in \Gamma(H)$ to the visual section $\mu \in \Gamma(V)$. The full space of graphics is not necessarily equivariant; therefore we formalize the constraints on Q such that it produces structure preserving graphics.

We formalize the expectation that visualization generation functions parameterized in the same way should generate the same functions as the equivariant map $Q : \mu \mapsto \rho$. We then define the constraint on Q such that if Q is applied to two visual sections μ and μ' that generate the same ρ then the output of μ and μ' acted on by the same monoid m must be the same. We do not define monoid actions on all of $\Gamma(H)$ because there may be graphics $\rho \in \Gamma(H)$ for which we cannot construct a valid mapping from V. Lets call the

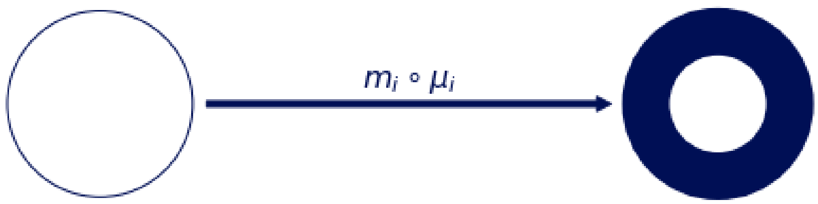


Figure 11: These two glyphs are generated by the same annulus Q function. The monoid action m_i on edge thickness μ_i of the first glyph yields the thicker edge $\mu_i{}'$ in the second glyph.

visual representations of the components $\Gamma(V) = X$ and the graphic $Q(\Gamma(V)) = Y$

Proposition 1. *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*

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \tag{23}$$

then a monoid action on Y can be defined as $m \circ \rho = \rho'$. If and only if Q satistfies Equation 23, we can state that the transformed graphic $\rho' = Q(m \circ \mu)$ is equivariant to a monoid action applied on Q with input $\mu \in Q^{-1}(\rho)$ that must generate valid ρ.

For example, given fiber $P = (xpos, ypos, color, thickness)$, 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 μ to $\mu' = (e, e, e, 3)$ and the corresponding action on ρ causes $Q(\mu')$ to be the thicker circle in Figure 11.

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

$$J = \{j \in K \text{ s. t. } \exists\gamma \text{ s.t. } \gamma(0) = k \text{ and } \gamma(1) = j\} \tag{24}$$

where the path[41] γ from k to j is a continuous function from the interval [0,1]. We define the glyph as the graphic generated by $Q(S_j)$

$$H \underset{\rho(S_j)}{\xleftarrow{}\!\!\!\xrightarrow{}} S_j \underset{\xi^{-1}(J)}{\overset{\xi(s)}{\xleftarrow{}\!\!\!\xrightarrow{}}} J_k \tag{25}$$

such that for every glyph there is at least one corresponding region on K, in keeping with the definition of glyph as any visually differentiable element put forth by Ziemkiewicz and Kosara[42]. The primitive point, line, and area marks[36, 43] are specially cased glyphs.

1.3.4 Assembly Q

Given the continuities described in 7, we illustrate a minimal Q that will generate the most minimal visualizations associated with those continuities: non-overlapping scatter points, a non-infinitely thin line, and an image.

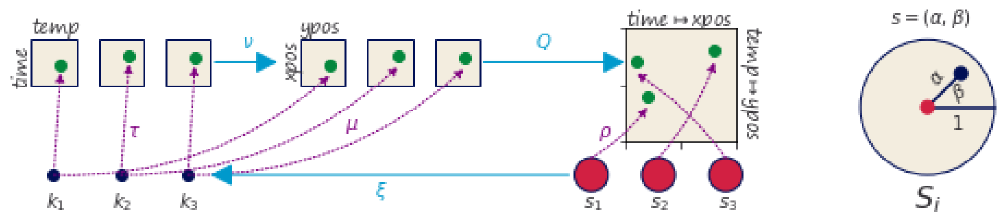


Figure 12: The data is discrete points (temperature, time). Via ν these are converted to (xpos, ypos) and pulled over discrete S. These values are then used to parameterize ρ which returns a color based on the parameters (xpos,ypos) and position α, β on S_k that ρ is evaluated on.

The scatter plot in Figure 12 can be defined as

$$Q(xpos, ypos)(\alpha, \beta) \tag{26}$$

with a constant *size* and color $\rho_{RGB} = (0, 0, 0)$ that are defined as part of Q. The position of this swatch of color can be computed relative to the location on the disc $(\alpha, \beta) \in S_k$ as shown in Figure 12

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

such that $\rho(s) = (x, y, 0, 0, 0)$ colors the point (x,y) black.

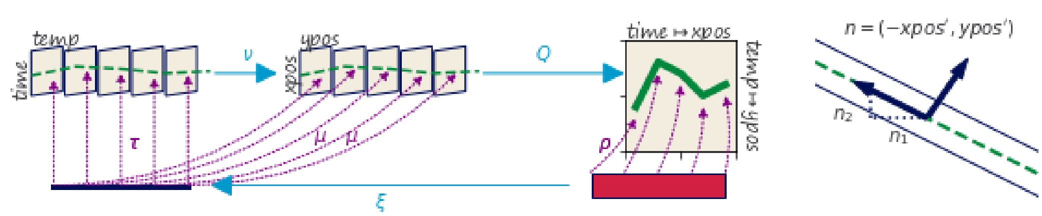


Figure 13: The line fiber ($time$, $temp$) is thickened with the derivative ($time'$, $temperature'$ because that information will be necessary to figure out the tangent to the point to draw a line. This is because the line needs to be pushed perpendicular to the tangent of (xpos, ypos). The data is converted to visual characteristics (xpos, ypos). The α coordinates on S specifies the position of the line, the β coordinate specifies thickness.

In contrast, the line plot

$$Q(xpos, \hat{n}_1, ypos, \hat{n}_2)(\alpha, \beta) \tag{27}$$

in **??** has a ξ function that is not only parameterized on k but also on the α distance along k and corresponding region in S. As 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; therefore the artist takes as input the jet bundle [44, 45] $\mathcal{J}^2(E)$ which is the data E and the first and second derivatives of E. 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 ρ

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

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.

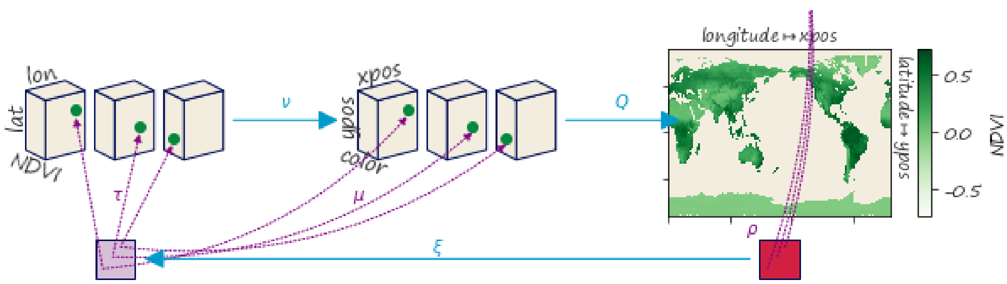


Figure 14: The only visual parameter an image requires is color since ξ encodes the mapping between position in data and position in graphic.

In Figure 14, the image

$$Q(xpos, ypos, color) \tag{28}$$

is a direct lookup into $\xi : S \to K$. The indexing variables (α, β) define the distance along the space, which is then used by ξ 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 ξ 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.

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 Equation 18, Q is a bundle map $Q : \xi^*V \to H$ where ξ^*V and H are both bundles over S.

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 μ

$$\xi^*V\ |_{\xi^{-1}(k)} = \xi^{-1}(k) \times P \tag{29}$$

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 Figure 15, given the section $\xi^*\mu$ pulled back from μ and the point $s \in \xi^{-1}(k)$, there is a direct map $(k, \mu(k)) \mapsto (s, \xi^*\mu(s))$ from μ over k to the section $\xi^*\mu$ over s. This

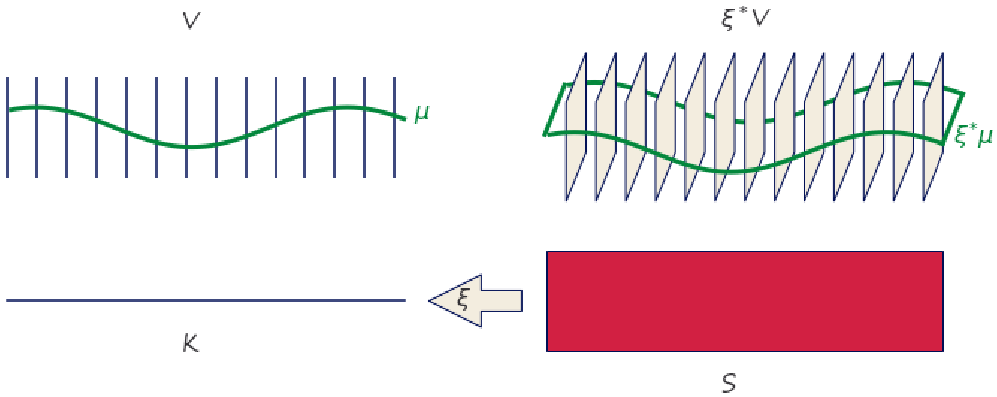


Figure 15: Because the pullback of the visual bundle ξ^*V is the replication of a μ over all points s that map back to a single k, we can constract a $\hat{Q}$ on μ over k that will fabricate the Q for the equivalent region of s associated to that k

means that the pulled back section $\xi^*\mu(s) = \xi^*(\mu(k))$ is the section μ copied over all $s$such that $\xi^*\mu$ is identical for all s where $\xi(s) = k$. In Figure 15 each dot on P is equivalent to the line on $P^*\mu$.

Given the equivalence between μ 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 ρ evaluated on a visual region s

$$\rho(s) \coloneqq Q(\xi^*\mu)(s) \tag{30}$$

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

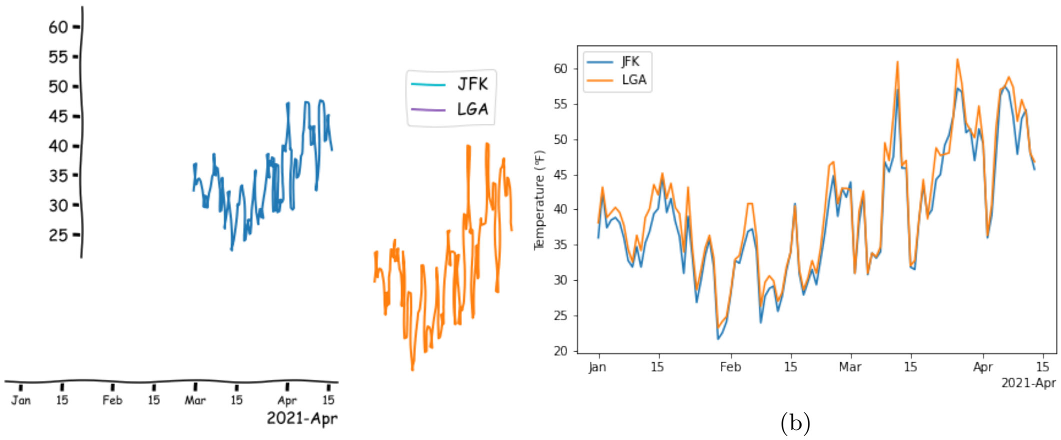
$$\hat{Q}(\mu(k))(s) \coloneqq Q((\xi^*\mu)(s)) \tag{31}$$

where $\hat{Q}$ with input μ 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 31 yields

$$\hat{Q}(\mu(k)) = Q(\xi^*\mu) \tag{32}$$

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 ρ to a back end such as Matplotlib.

1.3.5 Composition of Artists: $+$



(a) (b)

Figure 16: Each of the visual elements in **??** is generated via a unique artist A. In Figure 16a, they are added to the image independent of the other elements, creating an incoherant visualization. In Figure 16b, these artists are composited before being added to the image. Disjoint union of E aligns the two timeseries with the x and y axis so all these elements use a shared coordinate system. A more complex composition dictates that the legend is connected to the E such that it must use the same color as the data it is identifying.

Visualizations with a single artist do not provide much information, so we define addition operators for generating more complex visualizations. Given the family of artists $(E_i : i \in I)$ on the same image, the $+$ operator

$$+ \coloneqq \bigsqcup_{i \in I} E_i \tag{33}$$

defines a simple composition of artists. For example, the components in Figure 16a are each generated by different artists, and a visualization of solely the x axis is rarely all that useful. In Figure 16a, these artists are all added to the image independently of the other and therefore there are no constraints on how they are generated in conjunction with each other. In Figure 16b, the data is joined via disjoint union; doing so aligns the components in F such the ν to the same component in P targets the same coordinate system. 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 [46, 47] and brush-linked views[48, 49].

1.3.6 Equivalance class of artists A'

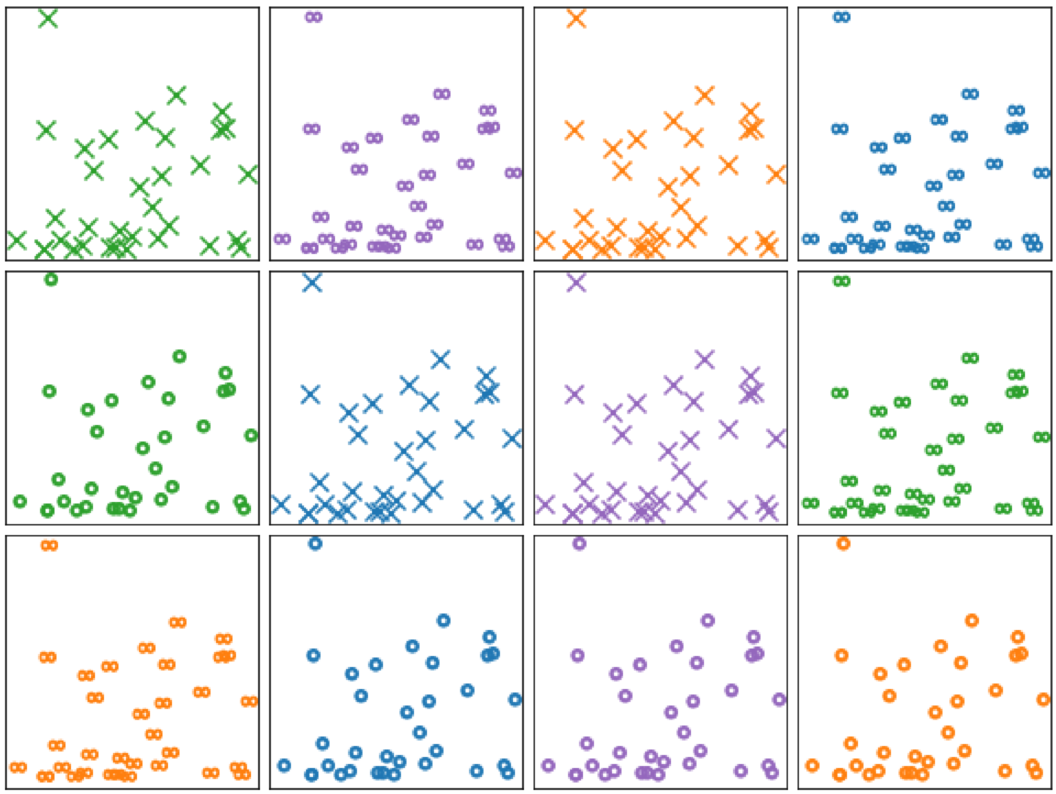


Figure 17: Each scatter plot is generated via a unique artist function A_i, but they only differ in aesthetic styling. Therefore, these artists are all members of an equivalence class $A_i \in A'$

Representational invariance, as defined by Kindlmann and Scheidegger, is the notion that visualizations are equivalent if changing the visual representation, such as colors or shapes, does not change the meaning of the visualization[50]. We propose that visualizations are invariant if they are generated by artists that are members of an equivalence class

$$\{A \in A' : A_1 \equiv A_2\}$$

For example, every scatter plot in Figure 17 is a scatter of the same datasets mapped to the *x position* and *y position* in the same way. The scatter plots only differ in the choice of constant visual literals, differing in color and marker shape. Each scatter is generated by an artist A_i, and every scatter is generated by a member of the equivalence class $A_i \in A'$. Since it is impractical to implement a new artist for every single graphic, the equivalance class provides a way to evaluate an implementation of a generalized artist. Given equivalent, but no necessarily identical, ν, Q, and ξ, two artists are equivalent. This criteria also allows for comparing artists across libraries.