1 Topological Equivariant Artist Model

To guide the implementation of structure preserving visualization components, we develop a mathematical formalism of visualization that specifies how these components preserve *continuity* and *equivariance*. Inspired by the analogous classes in Matplotlib [1], we call the transformation from data space to graphic space 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 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 model data (subsection 1.1) and graphics (subsection 1.2) as topological structures that encapsulate component types and continuity. We then discuss the functional 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

We use fiber bundles as the data model because they are inclusive enough to express all the types of structures of data described in **??**. 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. Our use of fiber bundles builds on Butler's work proposing that fiber bundles should be the common data abstraction for visualization data [2, 3]. 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 (subsubsection 1.1.4), 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 $K \times F$.

1.1.1 Variables in Fiber Space F

To formalize the structure of the data components, we use Spivak's description of the schema [6] as a fiber bundle to bind the components of the fiber to variable names and data types. 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 can also be written as

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

which allows us to decouple F into components $F_i = \mathbb{U}_{\sigma(c_i)}$.

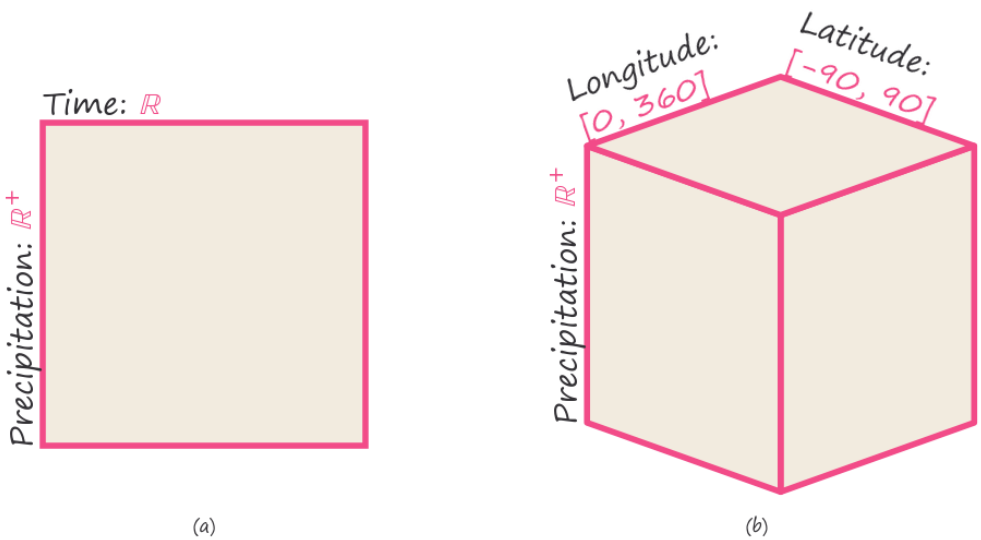


Figure 1: The fiber space is the cartesian product of the components. The 2D fiber $F = \mathbb{R} \times \mathbb{R}^+$ (a) encodes the properties of *time* and *precipitation* components. One dimension of the fiber encodes the range of possible values for the time component of the dataset, which is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

For example, the records in the 2D fiber (a) in Figure 1 are a pair of *times* and *precipitation* measurements taken at those times. Time is a positive number of type `datetime` which can be resolved to $\mathbb{U}_{\texttt{datetime}} = \mathbb{R}$. Precipitation 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 ($c = precipitation$, $T =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$) and is the set of values $\mathbb{U}_\sigma = \mathbb{R}$. In the 3D fiber (b) in Figure 1, 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

$$F = \mathbb{R} \times [0, 360] \times [-90, 90]$$

with components ($c = precipitation$, $T = \texttt{float}$, $\mathbb{U}_\sigma = \mathbb{R}$), ($c = latitude$, $T = \texttt{float}$, $\mathbb{U}_\sigma = [0, 360]$), and ($c = longitude$, $T = \texttt{float}$, $\mathbb{U}_\sigma = [-90, 90]$. By adapting Spivak's framework, our model has a consistent way to describe the components of the data, no matter their complexity.

1.1.2 Measurement Scales: Monoid Actions

Implementing expressive visual encodings requires formally describing the structure on the components of the fiber. We conjecture that the structure can be formally defined as the actions of a monoid on the component. We can also discuss complex structures since a core property of monoids is composability [7]. In doing so, we can specify the properties of the component that must be preserved in a graphic representation.

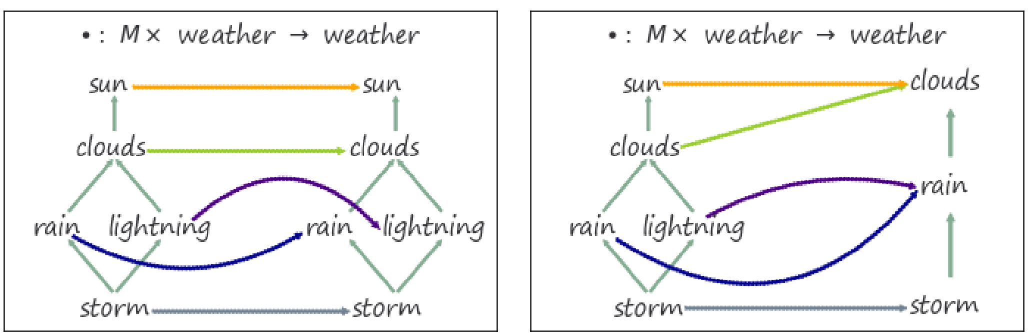
A monoid [8] M is a set with a binary operation $* : M \times M \to M$ that satisfies the axioms:

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

As defined on a component of F, a left monoid action [9, 10] of M_i is a set F_i with an action $\bullet : M \times F_i \to F_i$ with the properties:

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

We conjecture that given these properties, if there is a partial ordering on both sides of the action then the action is equivariant and monotonic.



(a) (b)

Figure 2: The action $\bullet$ in Figure 2a is the arrows from the partial order diagram of weather states on the left to the diagram of weather states on the right. The action maps the weather states to themselves, thereby preserving the ordering defined by the monoid $*$ on both sides of the action. We conjecture that the action in Figure 2b is monotonic when the partial ordering of the weather states is the same as the partial order of the elements they are mapped to. Given $sun \geq clouds \geq rain, lightning$ on the right, we conjecture that the action $sun, clouds \to clouds$, and $rain, lightning \to rain$ is structure preserving such that 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 Figure 2. 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 Figure 2, 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. 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 broad enough to include the Steven's measurement scales [12, 13].

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 added to the specification of the fiber $(c_i, T_i, \mathbb{U}_\sigma\, M_i)$

1.1.3 Continuity of the Data

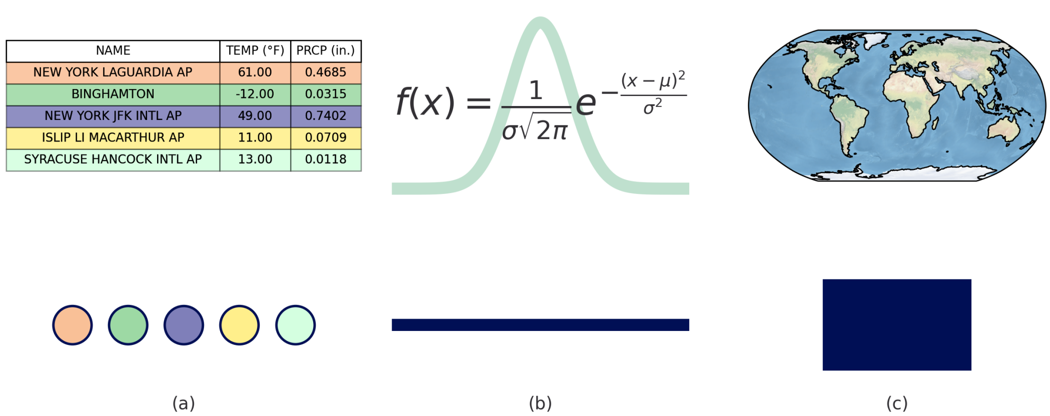


Figure 3: The topological base space K encodes the continuity of the data space. The table of discrete weather station records has discrete continuity such that each record maps to a single point (a). A gaussian has a value at all points along the interval x is sampled from and therefore has a 1D continuity (b). The globe has a value at all points (latitude, longitude) on the globe and therefore has 2D continuity (c).

The base space K provides a way to explicitly encode the continuity of the data, as described in ??. This explicit topology is a concise way of distinguishing between visualizations that appear identical but assume different continuity, for example heat maps and images. 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. Formally K is the quotient space [14] of E meaning it is the finest space [15] such that every $k \in K$ has a corresponding fiber F_k [14].

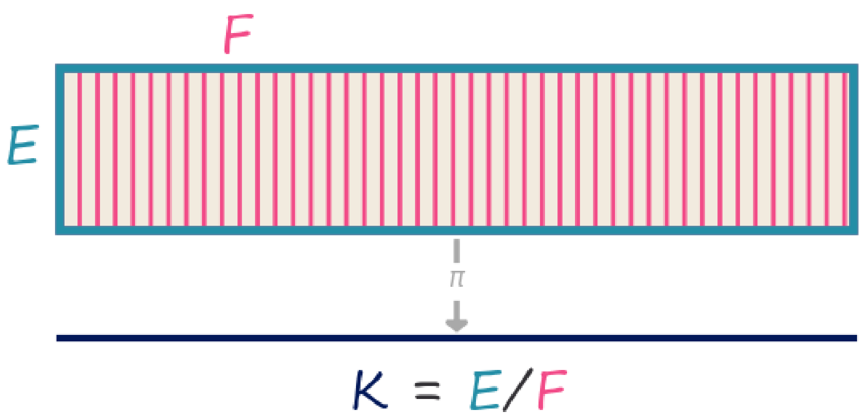


Figure 4: The total space E is divided into fiber segments F. The base space K acts as an index into the records in the fibers, such that every point k has a corresponding fiber F_k. The projection map π maps every fiber F_k to a point $k \in K$ in the base space.

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 $[0, 1]$. While the total space E may have components in F that describe any given point $k \in K$, such as *time, latitude, longitude*, these labels are indexed into from K the same as any other components. In contrast to the structural *keys* with associated *values* proposed by Munzner [16], our model treats keys k as a pure reference to topology. Decoupling the keys from their semantics allows the components identifying the keys to be altered, which provides for a coordinate agnostic representation of the continuity and facilitates encoding of data where the independent variable may not be clear. For example total rainfall is dependent on time of day and how much rain has already fallen; therefore changing the coordinate system should have no effect on how the records are connected to each other, as illustrated in **??** where precipitation in inches and millimeters yield equivalent line plots.

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 the monoid M_i acts on component bundle E_i. The K remains the same because the continuity of the data does not change just because there are fewer components in each record.

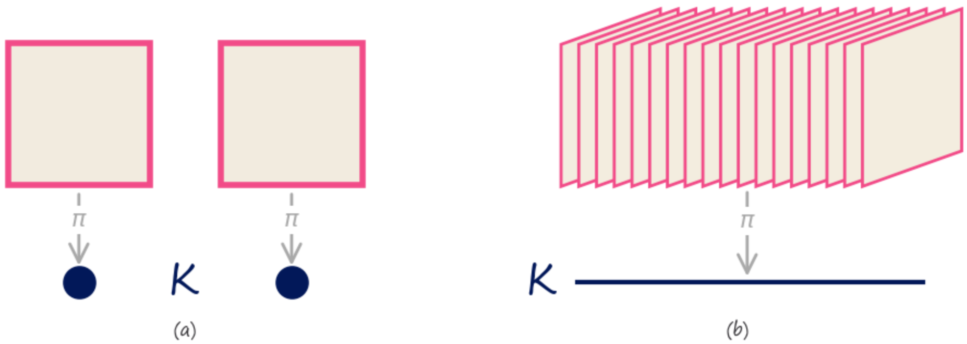


Figure 5: The fiber bundles in (a) and (b) encode the two component dataset from Figure 1, with *(time, precipitation)* components, as having different continuities. The fiber bundle with discrete continuity (a) encodes the dataset as being a set of discrete records. The fiber bundle over the continuous interval K (b) encodes the records as if they were sampled from a 1D continuous space.

The datasets in Figure 5 have the same fiber of (precipitation, time). The points (a) represent a discrete base space K, meaning that every dataset encoded in the fiber bundle has discrete continuity. The line (b) is a representation of a 1D continuity, meaning that every dataset in the fiber bundle is 1D continuous. Explicitly encoding data continuity, for example that (a) has discrete continuity and (b) is 1D continuous, provides a means to explicitly specify the continuities visualization components must preserve.

1.1.4 Data 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 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

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

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 can be decomposed as

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

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.

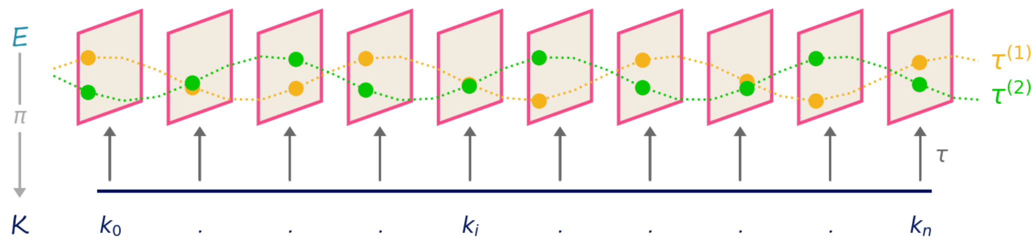


Figure 6: Fiber (time, precipitation) with a 1D continuous K defined on an interval $[0, n]$. The sections $\tau^{(1)}$ and $\tau^{(2)}$ are constrained such that the time variable must be monotonic, which means each section is a time series of precipitation values. They are included in the global set of sections $\tau^{(1)}, \tau^{(2)} \in \Gamma(E)$

In Figure 6, the fiber is the same encoding of (*time*, *precipitation*) 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, precipitation)* values. Section $\tau^{(2)}$ returns a different time series of *(time, precipitation)* values. Both sections are included in the global set of sections $\tau^{(1)}, \tau^{(2)} \in \Gamma(E)$.

1.1.5 Sheafs

Dynamic visualizations require evaluating sections on different subspaces of K; this can be achieved using a mathematical structure, called a sheaf $\mathcal{O}$, for defining collections of objects [17–19] 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 [20] and dynamically updated visualizations such as sliding windows [21, 22].

1.1.6 Applications

Using fiber bundles as the data abstraction allows the model to describe widely used data containers without sacrificing the semantic structure embedded in each container. For example, the section can be any instance of a numpy array [23] that stores an image, such as an image where the K is a 2D continuous plane and the F is $(\mathbb{R}^3, \mathbb{R}, \mathbb{R})$. In this fiber, the $\mathbb{R}^3$ components encode color, and the other two components are the x and y positions of the sampled data in the image. The continuity of the image is implicitly encoded in the array as the index, so the position components encode the resolution. Instead of an image, the numpy array could also store a 2D discrete table. The fiber may 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. Labeled containers can also be described in this framework because of the schema like structure of the fiber. One such example is a pandas series which stores a labeled list, another is a dataframe [24] which has the structure of a relational table. A series could store the values of $\tau^{(1)}$ and a second series could be $\tau^{(2)}$, while a dataframe would have multiple components and each data frame would be a unique section τ. The ability to encode complexity in continuity and components is particularly beneficial when working with N d imensional labeled data containers. For example, an xarray [25] data cube that stores precipitation would be a section of a fiber bundle with a K that is a continuous volume and components *(time, latitude, longitude, precipitation)*. This section does not need to resolve to values immediately and instead can be an instance of a distributed data container, such as a dask array [26].

1.2 Graphic Space H

To establish that the artist is a 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 τ, 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(subsubsection 1.2.3) 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 a non-pixelated 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

For a visualization component to preserve continuity, we propose that there must exist a structure preserving surjective map $\xi : S \to K$ from the data base space K to the graphic base space S. Formally, we require that K be a deformation retract [27] of S such that K

and S have the same homotopy, meaning there is a continuous map from S to K[28]. 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 $k \in K$. 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. The map ξ is part of the implementation of the artist $\mathcal{A}$ and therefore is not defined in terms of the data; instead it is how we specify the constraint that the type of the graphic *continuity* must be able to map to the type of the data *continuity*.

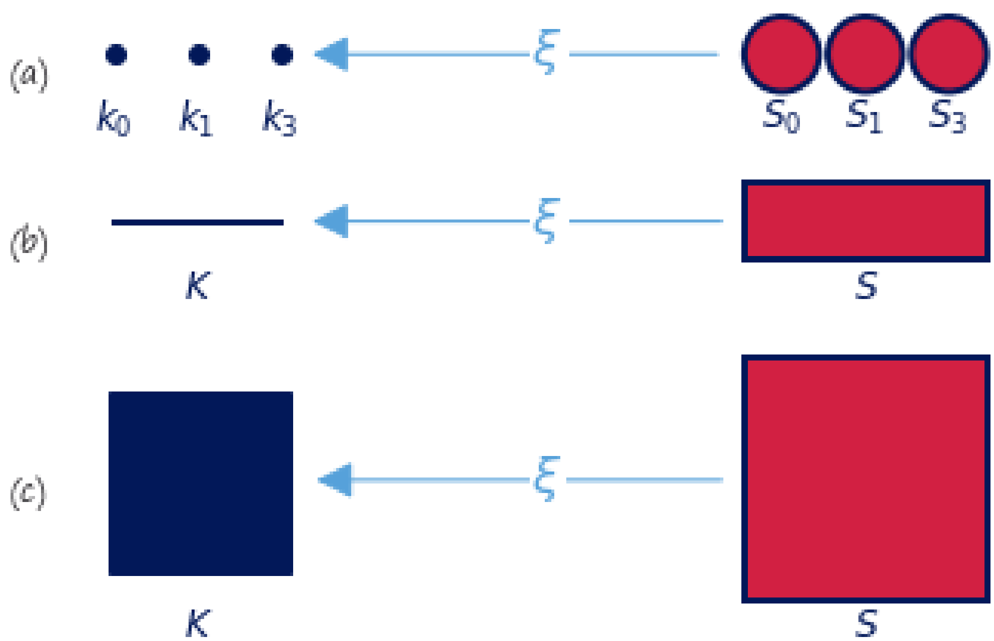


Figure 7: For a visualization component to preserve continuity, it must have a continuous surjective map $\xi : S \to K$ from graphic continuity to data continuity. The scatter (a) and line (b) graphic base spaces S have one more dimension of continuity than K so that S can encode physical aspects of the glyph, such as shape (a circle) or thickness. The image (c) has the same dimension in S as in K because K is already 2D and therefore can directly map into screen space.

To encode the continuity of the elements in the display fiber D, the graphic base space S has the same dimensionality as the target output space. For example, in Figure 7 the base space S is a representation of a region of a 2D display space. Since S must have the same dimensionality as the output graphic, it is allowed to add dimensions to K to make K renderable. A point that is 0D in K cannot be represented on screen unless it is thickened to

2D (a) to encode the connectivity of the pixels that visually represent the point. This is also the case with the line (b), which would be infinitely thin on screen if S was not thickened to 2D. 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 (c). Since the mapping function ξ binds the graphic base space to the data base space, it can be 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

The section $\rho : S \to H$ is the graphic in an idealized prerender space and also acts as a specification for rendering the graphic to target display format. To demonstrate the role of ρ 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 $S_p = {\rho_{x,y}}^{-1}(p)$ is a region $S_p \subset S$. The color is the result of the integration over S_p.

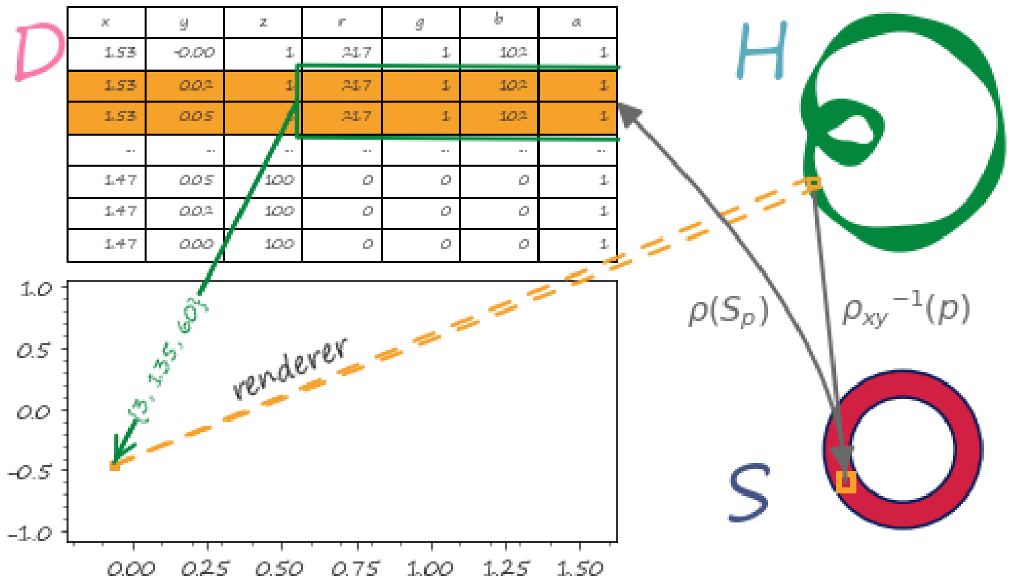


Figure 8: To render a graphic, a pixel p is selected in the display space, which is defined in the same coordinates as the x and y components in D via the renderer. Therefore, the pixel p maps to a region on H. In H the inverse mapping ${\rho_{x,y}}^{(}p)$ returns a region $S_p \subset S$. $\rho(S_p)$ 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 the color of the pixel. In general, ρ is an abstraction of rendering. In very broad strokes ρ can be a specification such as PDF [29], SVG [30], or an openGL scene graph [31] or a rendering engine such as cairo [32] or AGG [33]. Implementation of ρ is out of scope for this proposal.

1.3 Artist

We propose that visualization is structure preserving maps from data E to graphic H; having described E in subsection 1.1 and H in subsection 1.2, we now define the visual transformations from E to H that formalize the components that visualization libraries implement. The topological 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. We propose that the artist $\mathscr{A}$ is an *equivariant* map of monoid action $m \in M$

$$A(m \cdot r) = \varphi(m) \cdot A(r) \tag{16}$$

between data element $r \in \mathscr{E}$ and graphic element $A(r) \in \mathscr{H}$. To be equivariant with respect to monoids action, we conjecture that an artist carries a monoid homomorphism φ

$$\varphi : M \to M' \tag{17}$$

such that an action in data space $m \in M$ is equivalent to an action in graphic space $\varphi(M) \in M'$.

The artist A 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 18 facilitates 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. These graphic components are implicit visualization library APIs; by making them explicit as components of the fiber we can define expectations of how these parameters behave. As with the data and graphic bundles, the visual bundle (V, K, π, P) 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 [34], of the data section τ. The visual bundle V is the full design space [35] of possible parameters of a visualization type, such as a scatter plot or line plot. For example, one section μ of V is a tuple of visual values that specifies the visual characteristics of a part of a graphic.

$\boldsymbol{\nu_i}$	$\boldsymbol{\mu_i}$	$\boldsymbol{codomain(\nu_i) \subset P_i}$
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

In Table 1, the fiber components are specified by the visual parameter they are encoding. Multiple parameters can be encoded with the same transformation from data space to graphic space, for example x and y are both positions on a screen. Given a fiber of $\{x, y, color\}$ one possible section could be $\{.5, .5, (255, 20, 147)\}$. The $codomain(\nu_i)$ in Table 1 specifies the libraries internal representation of visual variables and can be used to determine which monoids can act on P_i.

1.3.2 Visual Encoders

We propose that the map from data components to graphic components $\nu : \tau \mapsto \mu$ is a monoid *equivariant* map. By specifying this constraints, we can guarantee that the stage of the artist that transforms data components into graphic representations is equivariant. These constraints then guide the implementation of reusable component transformers ν that are composed when generating the graphic. 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, then there is a monoid homomorphism from $\varphi : M_i \to {M_i}'$ 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. Therefore, a validly constructed ν is one where the diagram of the monoid transform m commutes

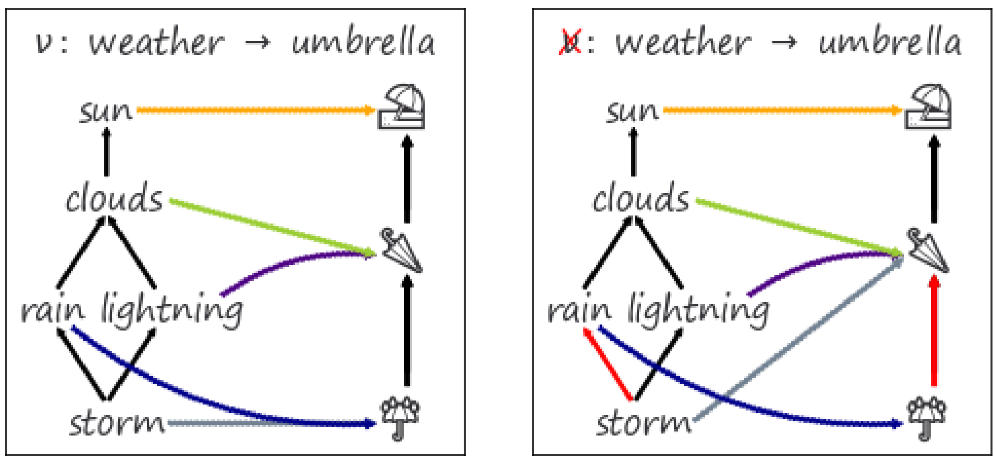
$$\begin{tikzcd} E_i \arrow[r, "\nu_i"] \arrow[d, "m_r"'] & V_i \arrow[d, "m_v"] \\ E_i \arrow[r, "\nu_i"] & V_i \end{tikzcd} \tag{21}$$

such that applying equivariant monoid actions to E_i and V_i preserves the map $\nu_i : E_i \to V_i$. 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, 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. We can make this assumption because 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 then 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 $(m, v) \to \varphi(m) \bullet v$, which lets us incorporate φ into the action $\bullet$ such that φ does not need to be explicitly defined in the constraints.



(a) (b)

Figure 9: The ν mapping in Figure 9a represented by the colored arrows is monotonic, and therefore monoid equivariant, since $\nu(storm) = \nu(rain)$ satisfies the condition $\nu(storm) \geq \nu(storm)$. In contrast, the map from data component to visual component in Figure 9b is not monotonic, and therefore not monoid equivariant, because $rain \geq storm$ is mapped to elements with the reverse ordering $\nu(storm) \geq \nu(storm)$.

The mapping from weather state to umbrella in Figure 9a is monotonic, and therefore we conjecture equivariant, because $\nu(rain) = \nu(storm)$ 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. In contrast, the translation from weather state data to visual representation as umbrella emoji in Figure 9b is an invalid visual encoding map ν because it is not monotonic and therefore not equivariant 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 9b would have to go in a different direction.

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: Equivariance constraints for the Stevens' measurement scales[36]

The Stevens measurement types [12], listed in Table 2, are specified in terms of groups, which are monoids with invertible operations[37]. We generalize to monoids to account for limitations in the types of data that can be described with the Stevens' scales [38, 39]

1.3.3 Visualization Assembly

Having described the maps to components in subsubsection 1.3.2, we now specify the assembly function $\hat{Q}$ that composites components in V into a graphic in H. Since the component transforms ν are equivariant, the equivariance constraints carry through to $\hat{Q}$. We specify these constraints to guide the implementation of library components responsible for generating graphics.

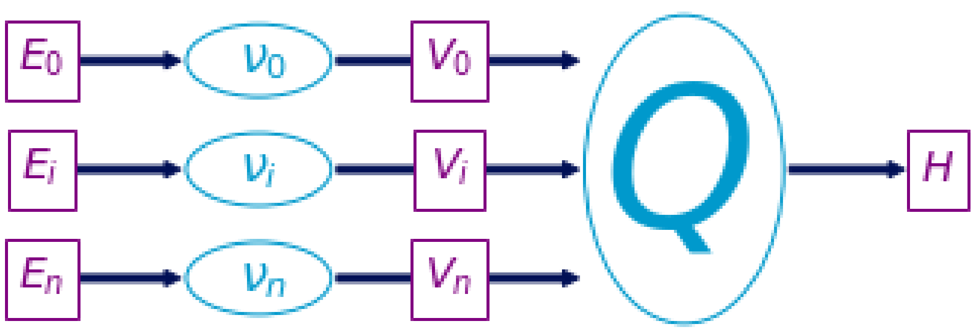


Figure 10: The transform functions ν_i convert data $\tau_i \in E$ to visual characteristics $\mu_i \in V$. These visual components μ_i are then assembled by Q into a graphic $\rho \in H$.

The transformation from data into graphic is analogous to a map-reduce operation; as illustrated in Figure 10, 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 define the visualization assembly function $Q : \mu \mapsto \rho$ as an equivariant map to formalize the expectation that two Q functions parameterized in the same way should generate the same graphic. 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

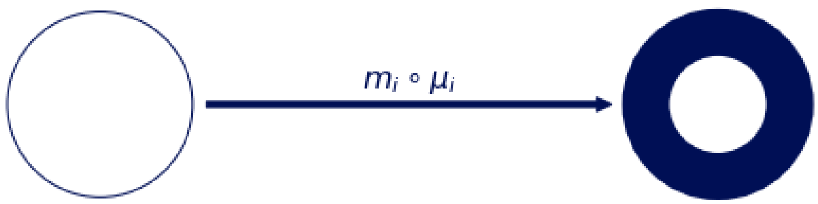


Figure 11: These two glyphs are generated by the same annulus Q function. The monoid action m_i on edge thickness μ_i of the first glyph yields the thicker edge $\mu_i{}'$ in the second glyph.

the 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 satisfies 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 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

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

where the path [40] γ from k to j is a continuous function from the interval [0,1]. We define the glyph as the graphic generated by $Q(S_j)$

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

such that for every glyph there is at least one corresponding region on K, in keeping with the definition of glyph as any visually distinguishable element put forth by Ziemkiewicz and Kosara [41]. The primitive point, line, and area marks [34, 42] 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.

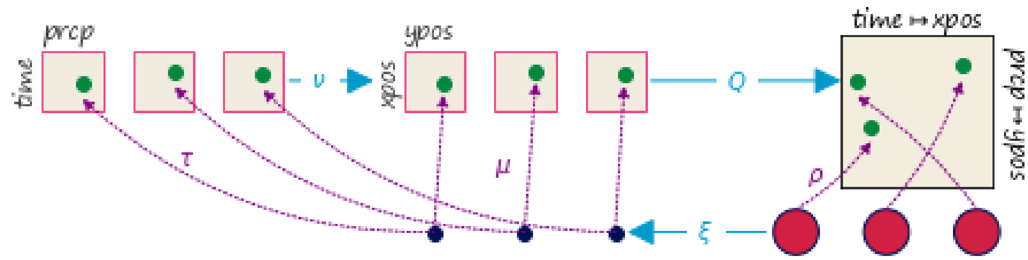


Figure 12: The data is discrete points (time, precipitation). Via ν these are converted to (xpos, ypos) and pulled over discrete S via ξ^*. The pulled back visual section ν is composited with the assembly function $\hat{Q} \circ \nu = \rho$ to produce the instructions to make the graphic ρ. The graphic section fills in the pixels in the screen via lookup on S.

The scatter plot in Figure 12 has a constant size and color $\rho_{RGB} = (0,0,0)$ that are defined as part of the point assembly function.

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

$$x = \text{size} * \alpha \cos(\beta) + xpos$$

$$y = \text{size} * \alpha \sin(\beta) + ypos$$

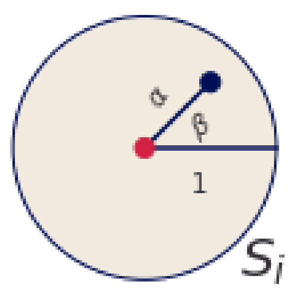


Figure 13: The simplest form of the scatter plot takes as input the expected position of the marker in visual space *(xpos, ypos)*. The marker shape is determined by the polar coordinates (α, β) on the disc; these coordinates dictate whether anything is drawn at that region of S. To obtain the color of the pixel at *(x,y)*, the region on S is scaled by a constant size and shifted by the *xpos* and *ypos*.

The position of this swatch of color is computed relative to the location on the disc $(\alpha, \beta) \in S_k$ as shown in Figure 13. The region $\alpha, \beta)$ is scaled by a constant size and shifted by *xpos* and *ypos*. This computation yields the values *(x,y)* that map into D and have a corresponding function $\rho(s) = (x, y, 0, 0, 0)$ which colors the point *(x,y)* black.

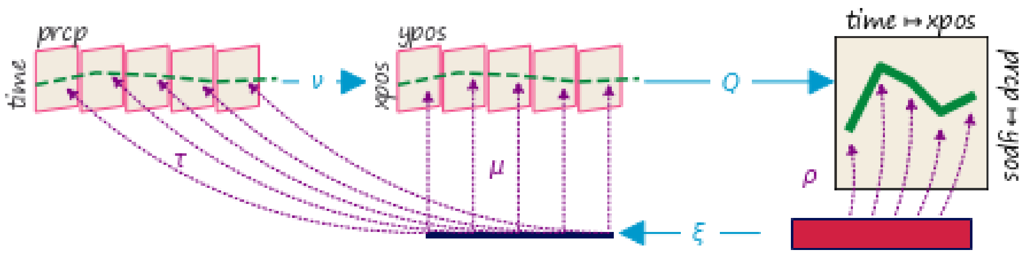


Figure 14: The line fiber $(time, precipitation)$ is thickened with the derivative $(time', precipitation'$ because that information will be necessary to figure out the tangent to the point to draw a line. This is because the line needs to be pushed perpendicular to the tangent of (xpos, ypos). The data 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 in Figure 14 has a ξ function that is not only parameterized on k but also on the α distance along the interval k and corresponding region in S.

$$Q(xpos, n_1, ypos, n_2)(\alpha, \beta) \tag{27}$$

$$|n| = \sqrt{{n_1}^2(\xi(\alpha)) + {n_2}^2(\xi(\alpha))}$$
$$\hat{n}_1 = \frac{n_1(\xi(\alpha))}{|n|}, \hat{n}_2 = \frac{n_2(\xi(\alpha))}{|n|}$$

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

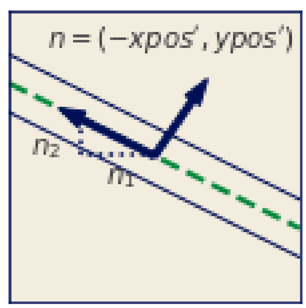


Figure 15: The $xpos$ and $ypos$ variables give the position of the line in screen space, but render an infinitely thin line. To draw equidistant lines parallel to($xpos$, $ypos$), defined by the distance (n_1, n_2), requires the derivatives $(n_1 = xpos', n_2 = ypos')$. The position ($xpos$, $ypos$) and width of the line is then used to determine whether a pixel is colored at the position (x, y). The values in data space are only looked up via the α coordinate of S because it maps to a location on K. The β parameter is used to specify how thick the line is in conjunction with the constant width.

As shown in Figure 15, 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 [43, 44] $\mathcal{J}^2(E)$ which is the data E and the first and second derivatives of E. The indexing map $\xi(\alpha)$ finds the point in K corresponding to the region in S at coordinate α. The section τ on the k that corresponds to the region in S returns the position *xpos, ypos* and the derivatives $\hat{n}_1, \hat{n}_2$. The derivatives are then multiplied by a width parameter to specify the thickness of the line. This is then used to determine the color of the pixel at *(x,y)*.

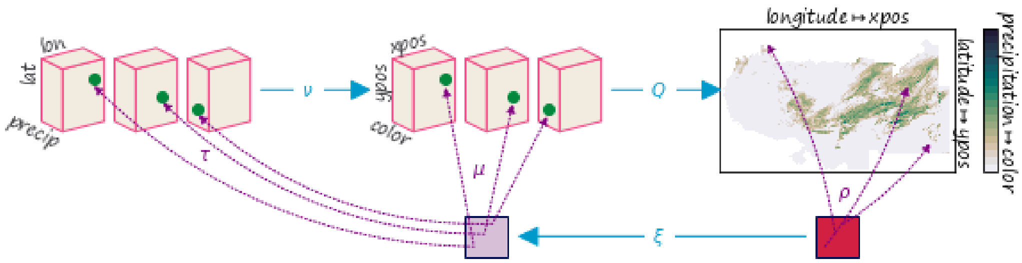


Figure 16: Via ξ the artist maps from a point *(x,y)* on the screen to a corresponding point on K. This maps into F via τ. These data points are converted to visual points via ν and then Q assembles the *(xpos, ypos, color)* parameters into attributes of each pixel.

In Figure 16, the image 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.

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

$$\begin{aligned} x &= \mathit{xpos}(\xi(\alpha)) \\ y &= \mathit{ypos}(\xi(\beta)) \\ R, G, B &= \mathit{color}(\xi(\alpha, \beta)) \end{aligned}$$

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

1.3.5 Assembly Template $\hat{Q}$

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.

$$\begin{tikzcd} E \arrow[r, "\nu"] \arrow[dr, "\pi"'] & V \arrow[d, shift right, "\pi"'] & \xi^*V \arrow[l, "\xi^*"'] \arrow[r, "Q"] \arrow[d, shift right, "\xi^*\pi"'] & H \arrow[dl, "\pi"] \\ & K \arrow[u, shift right, "\mu"'] & S \arrow[l, "\xi"] \arrow[u, shift right, "\xi^*\mu"'] & \end{tikzcd} \tag{29}$$

The map from graphic base space $\xi : S \to K$ (subsubsection 1.2.2) to data space maps many points in S to a single point in K. This means that 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

$$\xi^*V\mid_{\xi^{-1}(k)} = \xi^{-1}(k) \times P \tag{30}$$

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.

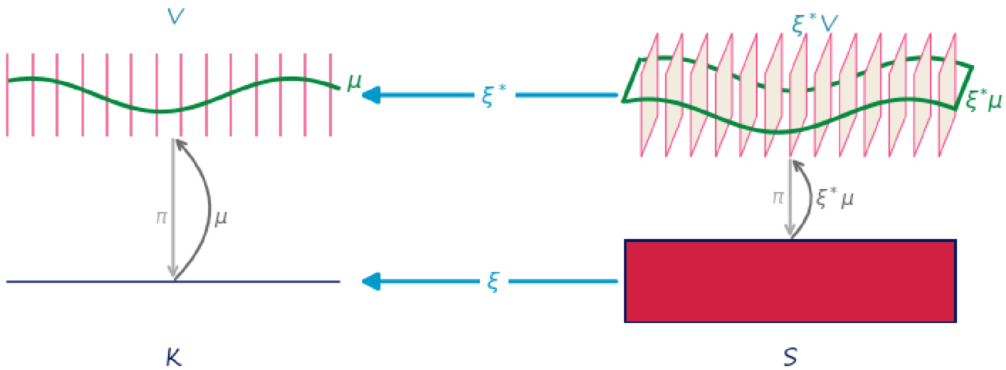


Figure 17: Because the pullback of the visual bundle ξ^*V is the replication of a μ over all points s that map back to a single k, we can construct a $\hat{Q}$ on μ over k that will fabricate the Q for the equivalent region of s associated to that k

As shown in Figure 17, given the section $\xi^*\mu$ pulled back from μ and the point $s \in \xi^{-1}(k)$, there is mapping from section $\xi^*\mu$ over s to μ over k. This 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 17 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

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

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

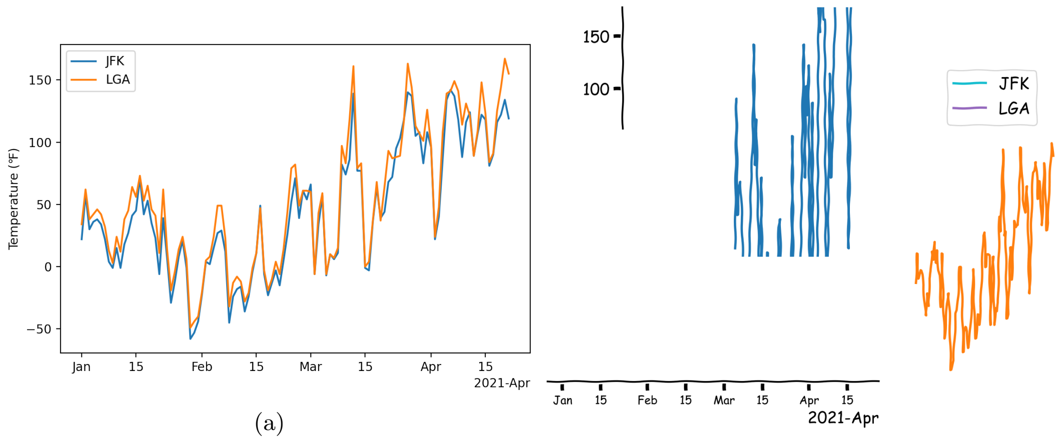
$$\hat{Q}(\mu(k))(s) \coloneqq Q((\xi^*\mu)(s)) \tag{32}$$

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. We can then factor s out of Equation 32, which yields

$$\hat{Q}(\mu(k)) = Q(\xi^*\mu) \tag{33}$$

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 restructure the existing pipeline for libraries that delegate the construction of ρ to a back end such as Matplotlib.

1.3.6 Composition of Artists: $+$



(a)

(b)

Figure 18: In Figure 18a, these artists are composited before being added to the image. Disjoint union of E aligns the two time series 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. None of this machinery exists in Figure 18b, therefore each artist is added to the page independent of the other elements.

Visualizations generally consist of more than one artist, commonly having visual elements such as the plot and axis labels and maybe legends. To generate these composite images, we define addition operators and specify the constraints for compositing artists. Given the family of artists $(E_i : i \in I)$ that are rendered to the same image, the $+$ operator

$$+ \coloneqq \bigsqcup_{i \in I} E_i \tag{34}$$

defines a simple composition of artists. For example, in Figure 18a 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. In Figure 18b, these artists are all added to the image independently of the other and therefore there are no constraints on where they are placed in the image. 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 [45, 46] and brush-linked views [47, 48].

1.3.7 Equivalance class of artists

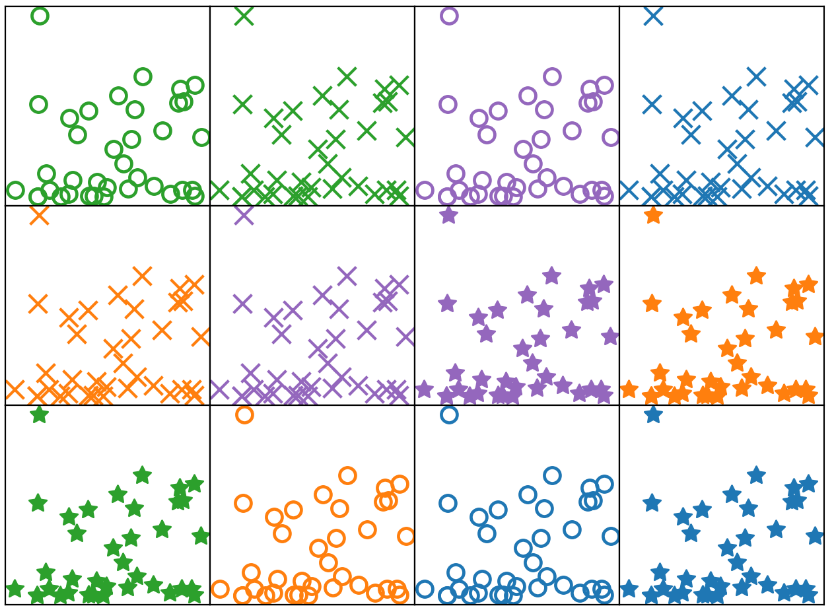


Figure 19: Each scatter plot is generated via a unique artist function A_i, but they only differ in aesthetic styling. Therefore, these artists are all members of an equivalence class $A_i \in A'$

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 [49]. By defining a criteria for invariance, we can evaluate whether two artists generate the same type of graphic and compare artists across libraries. 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 19 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 equivalence class provides a way to evaluate an implementation of a generalized artist.