

# 1 Topological Equivariant 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*.

$$\mathcal{A} : \mathcal{E} \rightarrow \mathcal{H} \quad (1)$$

The *artist*  $\mathcal{A}$  is a map from the data  $\mathcal{E}$  to graphic  $\mathcal{H}$  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 (subsection 1.2.2, data components to visual components (subsection 1.3.2), and visual components into graphic (subsection 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  $(E, K, \pi, F)$  defined by the projection map  $\pi$

$$F \hookrightarrow E \xrightarrow{\pi} K \quad (2)$$

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  $F$  (subsection 1.1.1), the continuity of records  $K$  (subsection 1.1.3), the collections of records  $\tau$  (??), 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  $\{T_0, \dots, T_m\} \in \mathbf{DT}$ , where  $\mathbf{DT}$  are the data types of the variables in the dataset. He then defines the single variable set  $\mathbb{U}_\sigma$

$$\begin{array}{ccc} \mathbb{U}_\sigma & \longrightarrow & \mathbb{U} \\ \pi_\sigma \downarrow & & \downarrow \pi \\ C & \xrightarrow{\sigma} & \mathbf{DT} \end{array} \quad (3)$$

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  $\sigma$ , the fiber for a one variable dataset is

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

where  $\sigma$  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:

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

which is equivalent to

$$F = F_0 \times \dots \times F_i \times \dots \times F_n \quad (6)$$

17 which allows us to decouple  $F$  into components  $F_i$ .

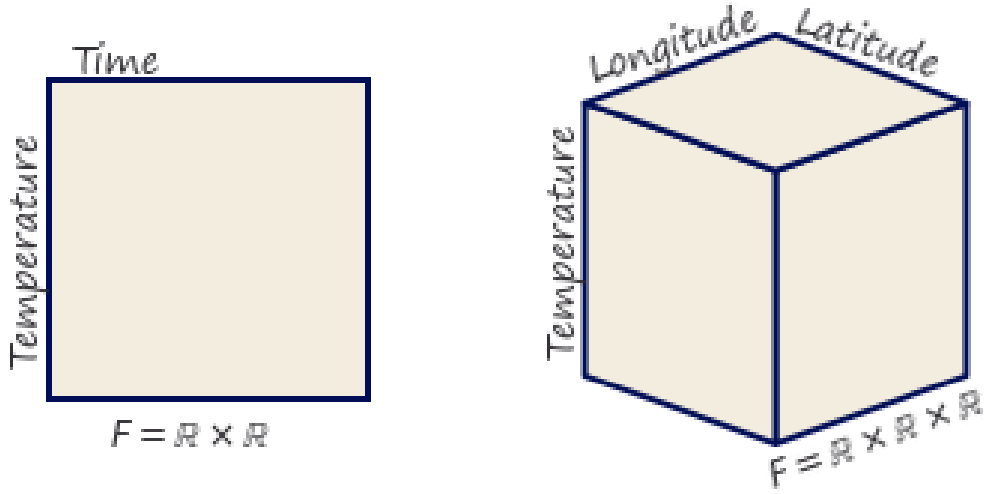


Figure 1: The fiber space is the set of all combinations of all theoretically possible values of the components. The 2D fiber  $F = \mathbb{R} \times \mathbb{R}$  encodes the properties of *time* and *temperature* components. One dimension of the fiber encodes the range of possible values for the time 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  $temperature \times time$ . 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  $\mathbb{U}_{\text{datetime}} = \mathbb{R}$ . Temperature values are real positive numbers  $\mathbb{U}_{\text{float}} = \mathbb{R}^+$ . The fiber is

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

where the first component  $F_0$  is the set of values specified by  $(c = \text{time}, T = \text{datetime}, \mathbb{U}_\sigma = \mathbb{R})$  and  $F_1$  is specified by  $(c = \text{temperature}, T = \text{float}, \mathbb{U}_\sigma = \mathbb{R})$  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

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

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

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

### 24 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:

**associativity** for all  $a, b, c \in M$   $(a \bullet b) \bullet c = a \bullet (b \bullet c)$

**identity** 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:

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

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

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

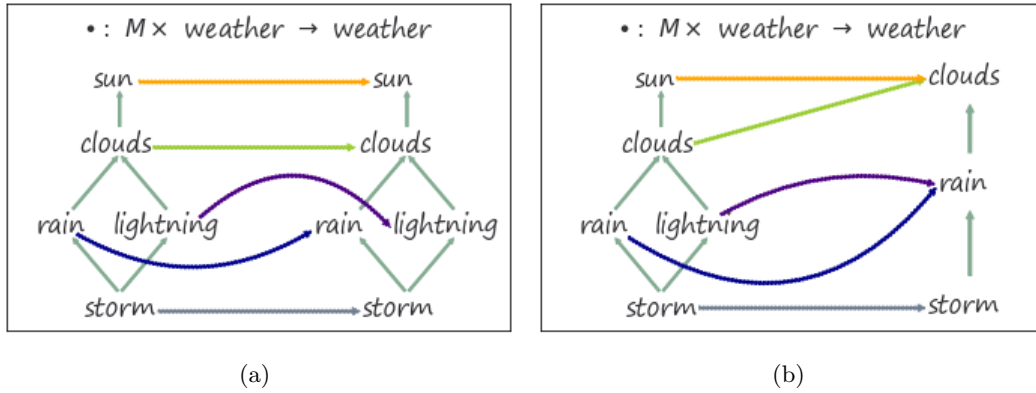


Figure 2: The action  $\bullet$  in ?? is the arrows from the partial order diagram of weather states on the left to the diagram of weather states on the right. Since the action maps the weather states to themselves, the ordering defined by the monoid  $*$  is preserved on both sides of the action. The 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  $\text{sun} \geq \text{clouds} \geq \text{rainlightning}$  on the right, the action  $\text{sun clouds} \rightarrow \text{clouds}$ , and  $\text{rainlightning} \rightarrow \text{rain}$  is structure preserving because on the left  $\text{cloud} \geq \text{rain}$  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

$$M = M_0 \times \dots \times M_i \times \dots \times \dots M_n \quad (7)$$

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*, *longititude*, 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.

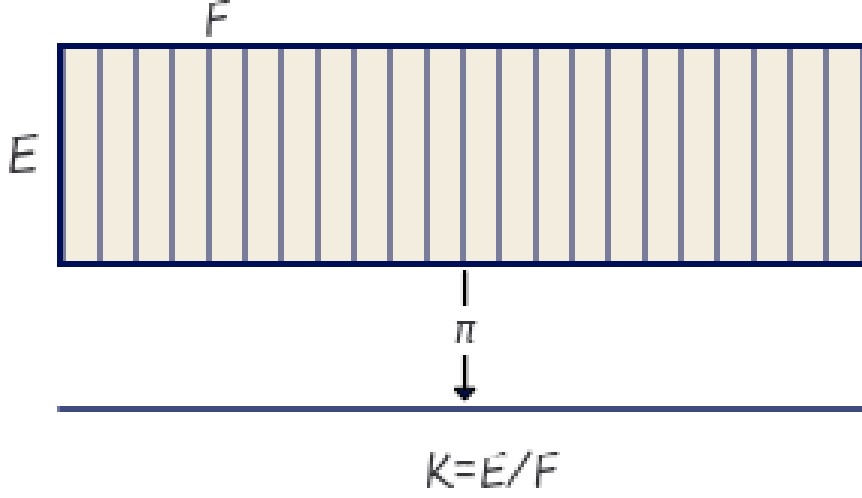


Figure 4: The base space  $E$  is divided into fiber segments  $F$ . The base space  $K$  acts as an index into the records in the fibers.

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

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

$$\pi : E_1 \oplus \dots \oplus E_i \oplus \dots \oplus E_n \rightarrow K \quad (8)$$

52 such that  $M_i$  acts on component bundle  $E_i$ . The  $K$  remains the same because the connec-  
 53 tivity of records does not change just because there are fewer components in each record. By  
 54 encoding this continuity in the model as  $K$  the data model now explicitly carries informa-  
 55 tion about its structure such that the implicit assumptions of the visualization algorithms  
 56 are now explicit.

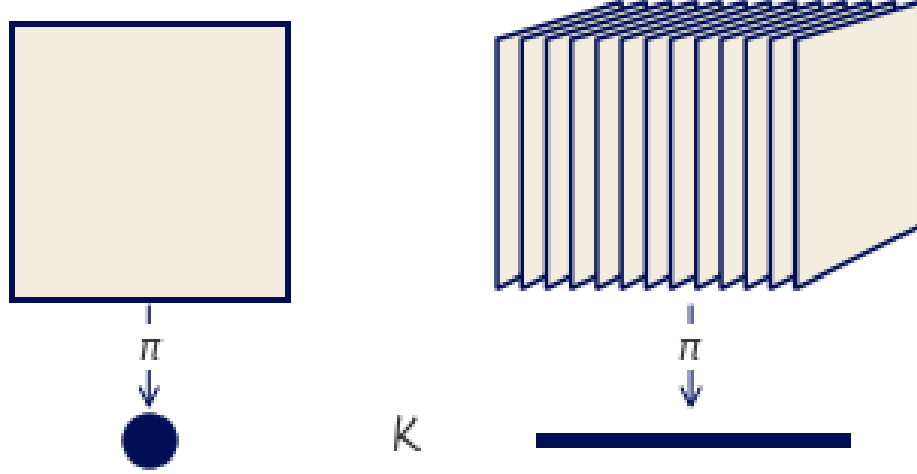


Figure 5: These two datasets have the same *(time, temperature)* fiber, but different continuities. The dataset on the left consists of discrete records, while the records in the dataset on the right sampled from a continuous space.

57 The datasets in [Figure 5](#) have the same fiber of (temperature, time). The dot represents  
 58 a discrete base space  $K$ , meaning that every dataset encoded in the fiber bundle has discrete  
 59 continuity. The line is a representation of a 1D continuity, meaning that every dataset in the  
 60 fiber bundle is 1D continuous. By encoding this continuity in the model as  $K$  the data model  
 61 now explicitly carries information about its structure such that the implicit assumptions of  
 62 the visualization algorithms are now explicit. The explicit topology is a concise way of  
 63 distinguishing visualizations that appear identical, for example heatmaps and images.

#### 64 1.1.4 Data $\tau$

While the projection function  $\pi : E \rightarrow K$  ties together the base space  $K$  with the fiber  $F$ , a section  $\tau : K \rightarrow E$  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  $\tau$  returns the record  $r$  at a given key in  $K$ . For any fiber bundle, there exists a map

$$\begin{array}{ccc} F & \hookrightarrow & E \\ & & \pi \downarrow \uparrow \tau \\ & & K \end{array} \quad (9)$$

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

$$\tau(k) = (k, (g_{F_0}(k), \dots, g_{F_n}(k))) \quad (10)$$

where  $g : K \rightarrow F$  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  $\tau$  as

$$\tau = (\tau_0, \dots, \tau_i, \dots, \tau_n) \quad (11)$$

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

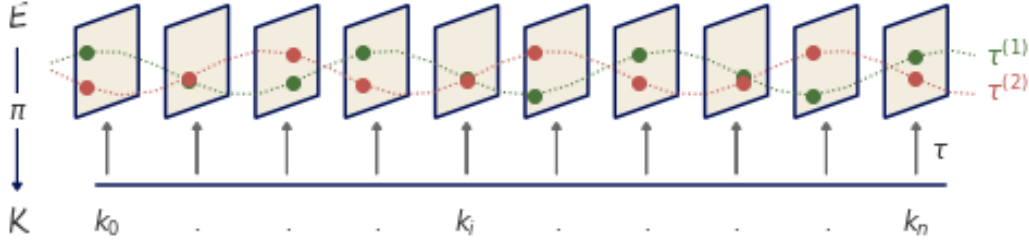


Figure 6: Fiber (time, temperature) with an interval  $K$  basespace. The sections  $\tau^{(1)}$  and  $\tau^{(2)}$  are constrained such that the time variable must be monotonic, which means each section is a timeseries of temperature values. They are included in the global set of sections  $\tau^{(1)},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

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  $\iota^*\tau$

$$\begin{array}{ccc} \iota^*E & \xhookrightarrow{\iota^*} & E \\ \pi \downarrow \uparrow \iota^*\tau & & \pi \downarrow \uparrow \tau \\ U & \xhookrightarrow{\iota} & K \end{array} \quad (12)$$

which are sections of  $E$  pulled back over local neighborhood  $U \subset E$  via the inclusion map  $\iota : E \rightarrow U$ . 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

81 instance of a univariate numpy array[24] that stores an image. This could be a section of a  
 82 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,  
 83 and the other two components are the x and y positions of the sampled data in the image.  
 84 This position information is already implicitly encoded in the array as the index and the  
 85 resolution of the image being stored. Instead of an image, the numpy array could also store a  
 86 2D discrete table. The fiber would not change, but the  $K$  would now be 0D discrete points.  
 87 These different choices in topology indicate, for example, what sorts of interpolation would  
 88 be appropriate when visualizing the data.

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

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

## 107 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  $\rho$  of a bundle  $(H, S, \pi, D)$ .

$$\begin{array}{ccc} D & \hookrightarrow & H \\ & & \pi \downarrow \uparrow \rho \\ & & S \end{array} \quad (13)$$

108 The graphic bundle  $H$  consists of a base  $S$ ([subsection 1.2.1](#)) that is a thickened form of  
 109  $K$  a fiber  $D$ ([subsection 1.2.2](#)) that is an idealized display space, and sections  $\rho$ ([??](#)) that  
 110 encode a graphic where the visual characteristics are fully specified.

### 111 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 $S$

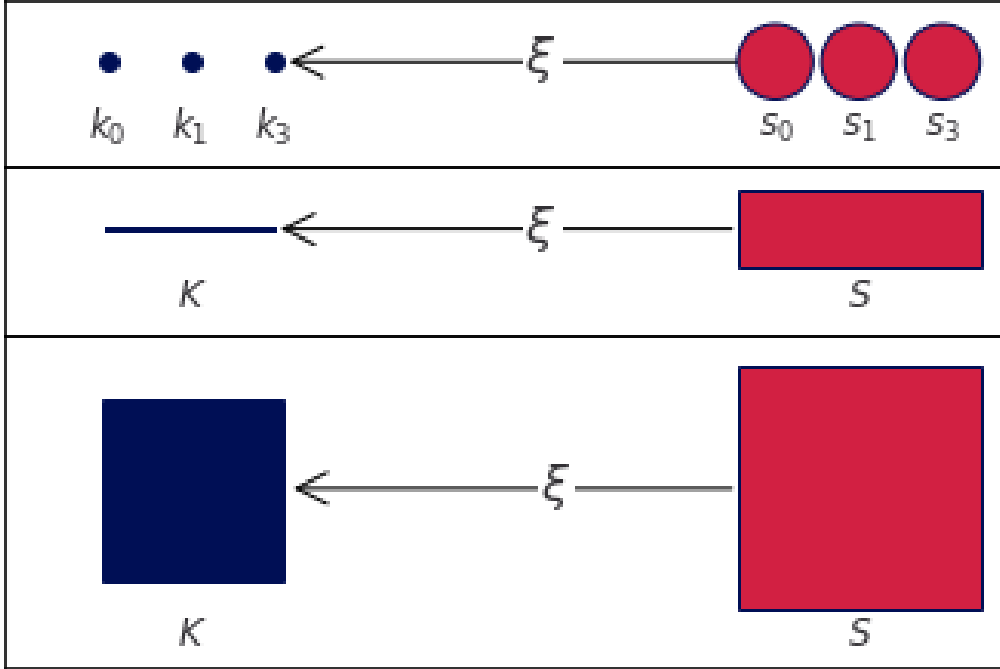


Figure 7: For a visualization component to preserve continuity, it must have a continuous surjective map  $\xi : S \rightarrow K$  from graphic continuity to data continuity. The scatter and line graphic base spaces  $S$  have one more dimension of continuity than  $K$  so that  $S$  can encode physical 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 there must be a continuous map  $\xi : S \rightarrow K$  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<sup>[28]</sup> of  $S$  so that  $K$  and  $S$  have the same homotopy, meaning there is a continuous map from  $S$  to  $K$ <sup>[29]</sup>. The surjective map

$$\xi : S \rightarrow K$$

$$\begin{array}{ccc} E & & H \\ \pi \downarrow & & \pi \downarrow \\ K & \xleftarrow{\xi} & S \end{array} \quad (14)$$

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  $\tau$  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  $\beta$  over a point  $\alpha_i$  specifies the thickness of the line in  $S$  for the corresponding  $\tau$  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  $\xi$  provides a way to identify the part of the visual transformation that is specific to the connectivity of the data rather than the values; for example it is common to flip a matrix when displaying an image. The  $\xi$  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 $\rho$

The section  $\rho : S \rightarrow H$  is the graphic in an idealized 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:

$$\begin{aligned} r_p &= \iint_{S_p} \rho_r(s) ds^2 \\ g_p &= \iint_{S_p} \rho_g(s) ds^2 \\ b_p &= \iint_{S_p} \rho_b(s) ds^2 \end{aligned}$$

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  $S_p$ .

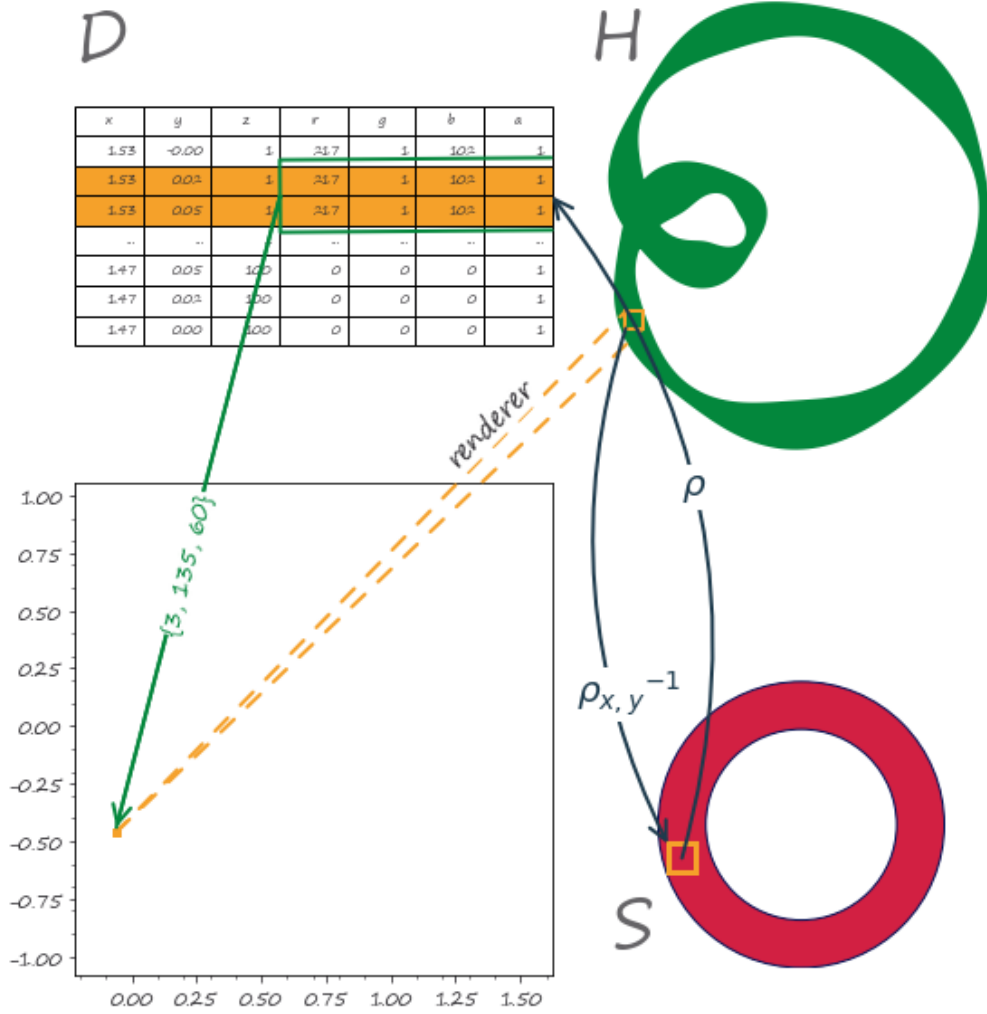


Figure 8: To render a graphic, a pixel  $p$  is selected in the display space, which is defined in the same coordinates as the  $x$  and  $y$  components in  $D$  via the renderer. In  $H$  the inverse mapping  $\rho_{xy}(p)$  returns a region  $S_p \subset S$ .  $\rho(S_p)$  returns a set of points  $(x, y, r, g, b) \in D$  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,  $\rho$  is an abstraction of rendering. In very broad strokes  $\rho$  can be a specification such as PDF[30], SVG[31], or an OpenGL

153 scene graph[32]. Alternatively,  $\rho$  can be a rendering engine such as cairo[33] or AGG[34].  
 154 Implementation of  $\rho$  is out of scope for this work,

### 155 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  $H$ ,  $\mathcal{O}(H)$ .

$$A : \mathcal{O}(E) \rightarrow \mathcal{O}(H) \quad (15)$$

The artist preserves *continuity* through the  $\xi$  map discussed in [subsubsection 1.2.2](#) and is an *equivariant* map because it carries a homomorphism of monoid actions [35]

$$\varphi : M \rightarrow M' \quad (16)$$

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

$$A(m \cdot r) = \varphi(m) \cdot A(r) \quad (17)$$

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

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

$$\begin{array}{ccccc} E' & \xrightarrow{\nu} & V & \xleftarrow{\xi^*} & \xi^*V & \xrightarrow{Q} & H \\ & \searrow \pi & \downarrow \pi & & \downarrow \xi^*\pi & \swarrow \pi & \\ & & K & \xleftarrow{\xi} & S & & \end{array} \quad (18)$$

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

#### 164 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  $(V, K, \pi, P)$  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  $\pi$

$$\begin{array}{c} P \hookrightarrow V \\ \pi \downarrow \uparrow \mu \\ K \end{array} \quad (19)$$

165 where  $\mu$  is the visual variable encoding, as described by Bertin [36], of the data section  $\tau$ .  
 166 The visual fiber  $P$  is defined in terms of the input parameters of the visualization library's

167 plotting functions; by making these parameters explicit components of the fiber, we can  
 168 build consistent definitions and expectations of how these parameters behave.

$\nu_i$	$\mu_i$	$\text{codomain}(\nu_i) \subset P_i$
position	x, y, z, theta, r	$\mathbb{R}$
size	linewidth, markersize	$\mathbb{R}^+$
shape	markerstyle	$\{f_0, \dots, f_n\}$
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

169 A section  $\mu$  is a tuple of visual values that specifies the visual characteristics of a part  
 170 of the graphic. For example, given a fiber of  $\{x, y, \text{color}\}$  one possible section could be  
 171  $\{.5, .5, (255, 20, 147)\}$ . The  $\text{codomain}(\nu_i)$  determines which monoids can act on  $P_i$ . These  
 172 fiber components are implicit in the library, as seen in [Table 1](#), and by making them explicit  
 173 as components of the fiber we can build consistent definitions and expectations of how these  
 174 parameters behave.

### 175 1.3.2 Visual Encoders $\nu$

We define the visual transformers  $\nu$

$$\{\nu_0, \dots, \nu_n\} : \{\tau_0, \dots, \tau_n\} \mapsto \{\mu_0, \dots, \mu_n\} \quad (20)$$

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  $\nu$  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  $\nu$  is one where the diagram of the monoid transform  $m$  commutes

$$\begin{array}{ccc} E_i & \xrightarrow{\nu_i} & V_i \\ m_r \downarrow & & \downarrow m_v \\ E_i & \xrightarrow{\nu_i} & V_i \end{array} \quad (21)$$

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  $\nu_i$  must satisfy the constraint

$$\nu_i(m_r(E_i)) = \varphi(m_r)(\nu_i(E_i)) \quad (22)$$

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

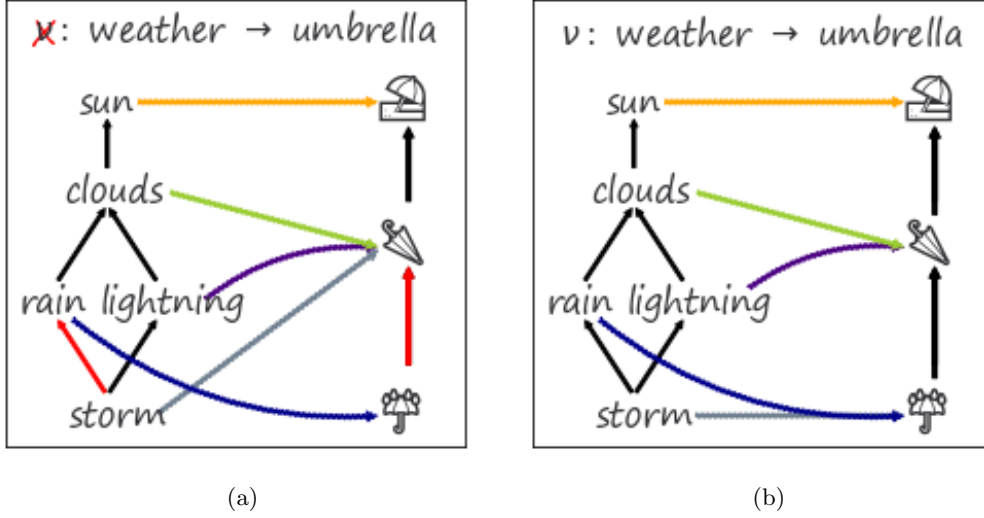


Figure 9: The map from data component to visual component in Figure 9a is not homomorphic, and therefore invalid, because  $rain \geq storm$  is mapped to elements with the reverse ordering  $v(storm) \geq v(rain)$ . In contrast, the mapping in Figure 9b is valid since  $v(storm) = v(rain)$  satisfies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

182 The translation from weather state data to visual representation as umbrella emoji in  
 183 Figure 9a is an invalid visual encoding map  $\nu$  because it is not homomorphic. This is  
 184 because the monotonic condition  $rain \geq storm \implies \nu(rain) \geq \nu(storm)$  is not met since  
 185  $\nu(rain) \leq \nu(storm)$ . To satisfy the monotonic condition for  $rain \geq storm$ , either red arrow  
 186 in Figure 9a would have to go in a different direction. On the other hand, the mapping  
 187 from weather state to umbrella in Figure 9b is a homomorphism since  $\nu(rain) = \nu(storm)$   
 188 satisfies the monotonic condition of  $rain \geq storm$ . Figure 9 is an example of how the  
 189 model supports partially ordered data components, which was a motivation for defining  
 190 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

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

### 198 1.3.3 Visualization Assembly

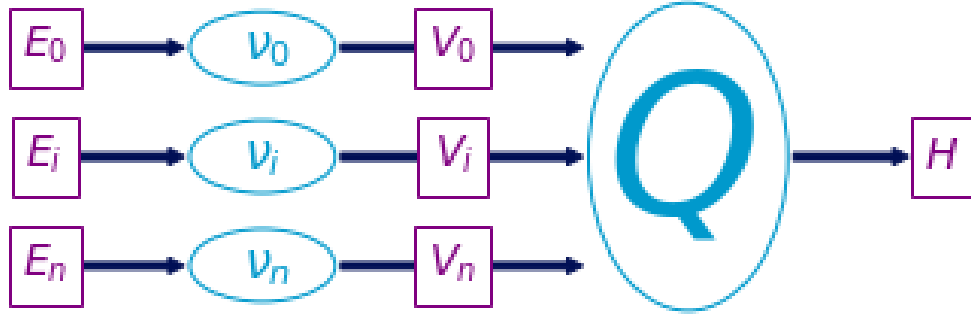


Figure 10: The transform functions  $\nu_i$  convert data  $\tau_i \in E$  to visual characteristics  $\mu_i \in V$ , then  $Q$  assembles  $\mu_i$  into a graphic  $\rho \in H$ .

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

205 We formalize the expectation that visualization generation functions parameterized in  
 206 the same way should generate the same functions as the equivariant map  $Q : \mu \mapsto \rho$ . We  
 207 then define the constraint on  $Q$  such that if  $Q$  is applied to two visual sections  $\mu$  and  $\mu'$   
 208 that generate the same  $\rho$  then the output of  $\mu$  and  $\mu'$  acted on by the same monoid  $m$   
 209 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

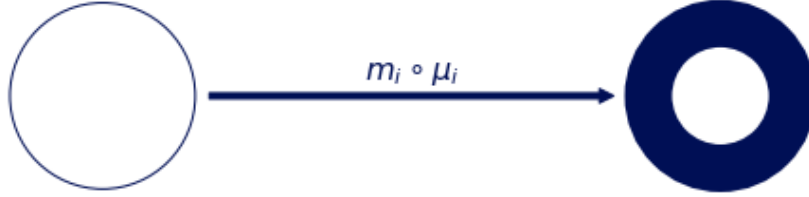


Figure 11: These two glyphs are generated by the same annulus  $Q$  function. The monoid action  $m_i$  on edge thickness  $\mu_i$  of the first glyph yields the thicker edge  $\mu_i'$  in the second glyph.

210 visual representations of the components  $\Gamma(V) = X$  and the graphic  $Q(\Gamma(V)) = Y$   
 211

**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*

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \quad (23)$$

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

215 For example, given fiber  $P = (xpos, ypos, color, thickness)$ , then sections  $\mu = (0, 0, 0, 1)$   
 216 and  $Q(\mu) = \rho$  generates a piece of the thin hollow circle. The action  $m = (e, e, e, x + 2)$ ,  
 217 where  $e$  is identity, translates  $\mu$  to  $\mu' = (e, e, e, 3)$  and the corresponding action on  $\rho$  causes  
 218  $Q(\mu')$  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

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

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

$$H \xrightleftharpoons[\rho(S_j)]{} S_j \xrightleftharpoons[\xi^{-1}(J)]{\xi(s)} J_k \quad (25)$$

219 such that for every glyph there is at least one corresponding region on  $K$ , in keeping with  
 220 the definition of glyph as any visually differentiable element put forth by Ziemkiewicz and  
 221 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.

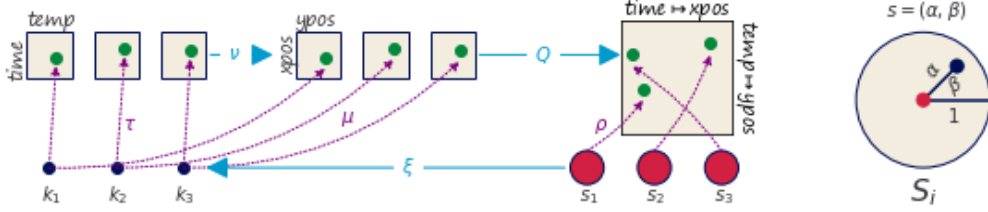


Figure 12: The data is discrete points (temperature, time). Via  $\nu$  these are converted to (xpos, ypos) and pulled over discrete  $S$ . These values are then used to parameterize  $\rho$  which returns a color based on the parameters (xpos,ypos) and position  $\alpha, \beta$  on  $S_k$  that  $\rho$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

The scatter plot in Figure 12 can be defined as

$$Q(xpos, ypos)(\alpha, \beta) \quad (26)$$

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.

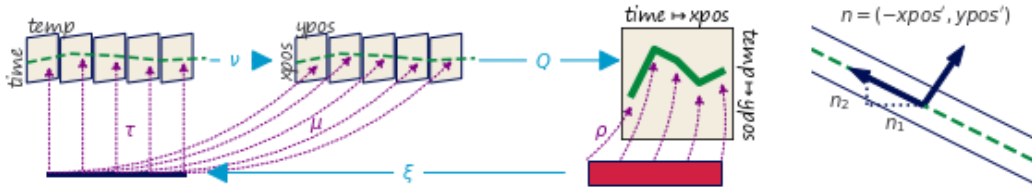


Figure 13: The line fiber  $(time, temp)$  is thickened with the derivative  $(time', temperature')$  because that information will be necessary to figure out the tangent to the point to draw a line. This is because the line needs to be pushed perpendicular to the tangent of (xpos, ypos). The data is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

In contrast, the line plot

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

in ?? has a  $\xi$  function that is not only parameterized on  $k$  but also on the  $\alpha$  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  $\rho$

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

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

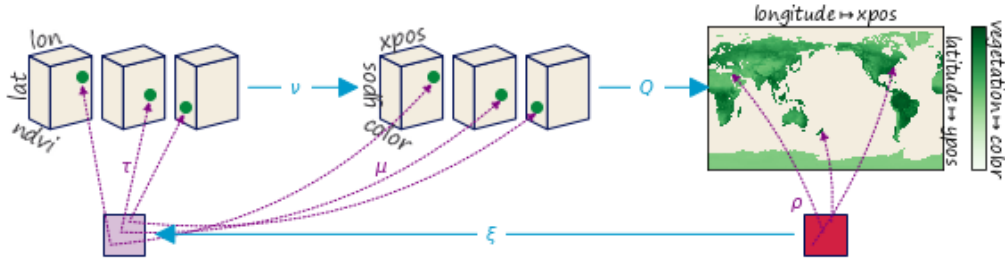


Figure 14: The only visual parameter an image requires is color since  $\xi$  encodes the mapping between position in data and position in graphic.

In Figure 14, the image

$$Q(xpos, ypos, color) \quad (28)$$

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

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

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

232 The graphic base space  $S$  is not accessible in many architectures, including Matplotlib;  
233 instead we can construct a factory function  $\hat{Q}$  over  $K$  that can build a  $Q$ . As shown in  
234 Equation 18,  $Q$  is a bundle map  $Q : \xi^*V \rightarrow H$  where  $\xi^*V$  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  $\mu$

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

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

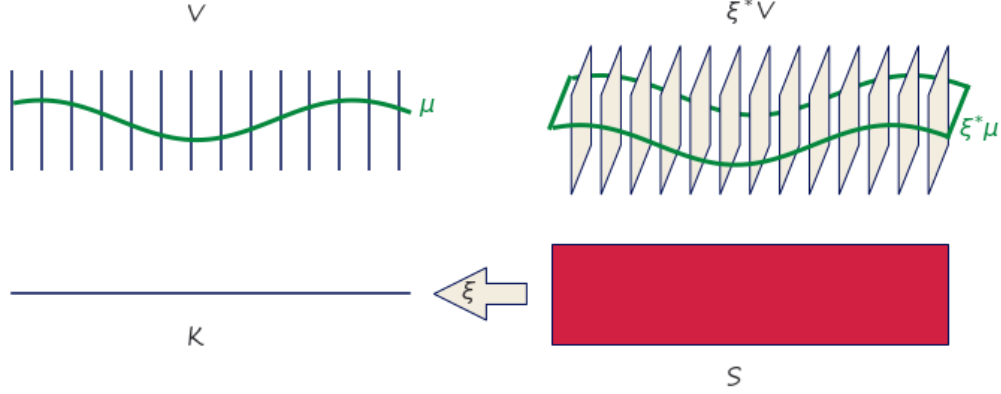


Figure 15: Because the pullback of the visual bundle  $\xi^*V$  is the replication of a  $\mu$  over all points  $s$  that map back to a single  $k$ , we can construct a  $\hat{Q}$  on  $\mu$  over  $k$  that will fabricate the  $Q$  for the equivalent region of  $s$  associated to that  $k$

As shown in **Figure 15**, given the section  $\xi^*\mu$  pulled back from  $\mu$  and the point  $s \in \xi^{-1}(k)$ , there is a direct map  $(k, \mu(k)) \mapsto (s, \xi^*\mu(s))$  from  $\mu$  over  $k$  to the section  $\xi^*\mu$  over  $s$ . This means that the pulled back section  $\xi^*\mu(s) = \xi^*(\mu(k))$  is the section  $\mu$  copied over all  $s$  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  $\mu$  and  $\xi^*\mu$  defined above, the reliance on  $S$  can be factored out. When  $Q$  maps visual sections into graphics  $Q : \Gamma(\xi^*V) \rightarrow \Gamma(H)$ , if we restrict  $Q$  input to  $\xi^*\mu$  then the graphic section  $\rho$  evaluated on a visual region  $s$

$$\rho(s) := Q(\xi^*\mu)(s) \quad (30)$$

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

$$\hat{Q}(\mu(k))(s) := Q((\xi^*\mu)(s)) \quad (31)$$

where  $\hat{Q}$  with input  $\mu$  is defined to  $Q$  that takes as input the copied section  $\xi^*\mu$  such that both functions are evaluated over the same location  $\xi^{-1}(k) = s$  in the base space  $S$ . Factoring out  $s$  from **Equation 31** yields

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

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  $\hat{Q} : \Gamma(V) \rightarrow \Gamma(H)$  locally over  $k$  such that it can be evaluated on a single visual record  $\hat{Q} : \Gamma(V_k) \rightarrow \Gamma(H|_{\xi^{-1}(k)})$ . This allows us to construct a  $\hat{Q}$  that only depends on  $K$ , such that for each  $\mu(k)$  there is part of  $\rho|_{\xi^{-1}(k)}$ . The construction of  $\hat{Q}$  allows us to retain the functional map reduce benefits of  $Q$  without having to majorly restructure the existing pipeline for libraries that delegate the construction of  $\rho$  to a back end such as Matplotlib.

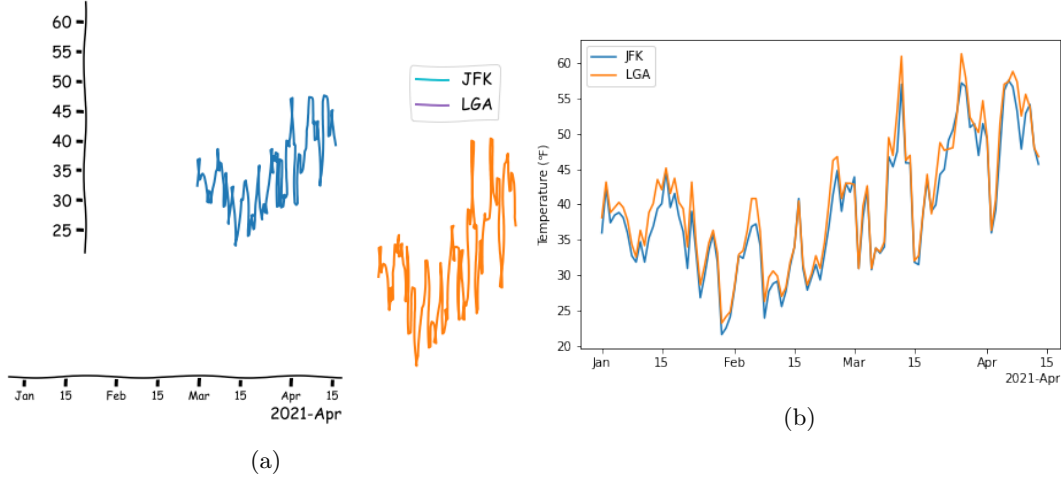


Figure 16: Each of the visual elements in ?? is generated via a unique artist  $A$ . In Figure 16a, they are added to the image independent of the other elements, creating an incoherent visualization. In Figure 16b, these artists are composited before being added to the image. Disjoint union of  $E$  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.

We describe the constraints for compositing artists by defining addition operators. Given the family of artists  $(E_i : i \in I)$  that are rendered to the same image, the  $+$  operator

$$+ := \sqcup_{i \in I} E_i \quad (33)$$

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  $\nu$  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 Equivalence class of artists $A'$

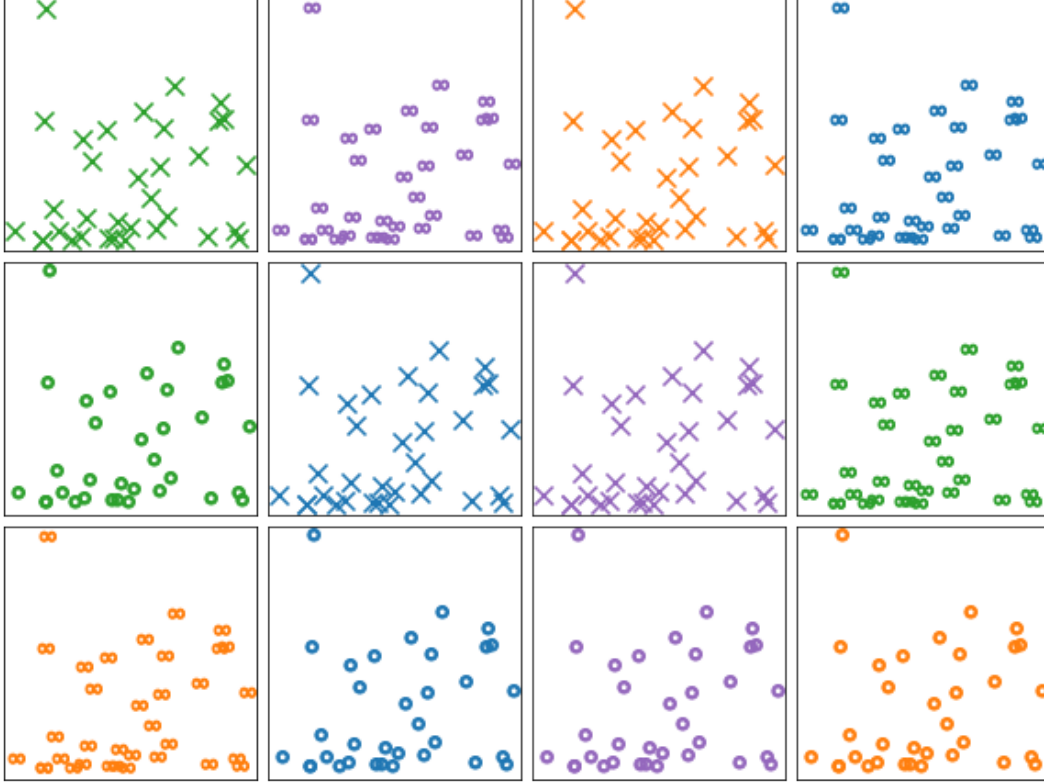


Figure 17: Each scatter plot is generated via a unique artist function  $A_i$ , but they only differ in aesthetic styling. Therefore, these artists are all members of an equivalence class  $A_i \in A'$

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 equivalence class provides a way to evaluate an implementation of a generalized artist. Given equivalent, but not necessarily identical,  $\nu$ ,  $Q$ , and  $\xi$ , two artists are equivalent. This criteria also allows for comparing artists across libraries.