# Topological Equivariant Artist Model for Visualization Library Architecture

Hannah Aizenman, Mikael Vejdemo-Johansson, Thomas Caswell, and Michael Grossberg, *Member, IEEE,*

## I. Introduction

Visualization design guidelines, generally, describe how to choose visual encodings that preserve the structure of the data; to follow these guidelines the visualization tools that implement these $\texttt{data} \rightarrow \texttt{graphic}$ transforms must be structure preserving. Loosely, preserving structure means that the properties of the data and how the points are connected to each other should be inferable from the graphic such that a $\texttt{graphic} \rightarrow \texttt{data}$ mapping can be made. For example, values read off a bar chart have to be equivalent to the values used to construct that chart. Therefore a visualization tool is structure preserving when it preserves the bidirectional mapping $\texttt{data} \leftrightarrow \texttt{graphic}$.

We propose that we can better enforce this expectation in software by providing a uniform way of expressing $\texttt{data}$ and $\texttt{graphic}$ using their respective algebraic structure and by uniformly specifying the behaviors and properties of those structures and the maps between them using category theory. For example, our framework can encapsulate how a table and scatter plot and heatmap are different representations of the same data and track an observation from a data cube as a point along a time series and on a map and in a network. The algebraic structures can then be translated into programmatic types, while the categorical descriptions translate to a functional design framework. Strong typing and function composition enable visualization software developers to build complex components from simpler verifiable parts [1], [2]. These components can be built as a standalone library and integrated into existing libraries and we hope these ideas will influence the architecture of critical data visualization libraries, such as Matplotlib.

The contribution of this paper is a methodology for describing structure, verifying structure preservation, and specifying the conditions for constructing a structure preserving map between data and graphics. This framework also provides guidance for the construction and testing of structure preserving visualization library components.

H. Aizenman is with the department of Computer Science, The Gradaute Center, CUNY.
E-mail: haizenman@gradcenter.cuny.edu,
M. Grossberg is with the department of Computer Science, City College of New York, CUNY. E-mail: mgrossberg@ccny.cuny.edu
Mikael Vejdemo-Johansson is with the department of Mathematics, CUNY College of Staten Island.
E-mail: mvj@math.csi.cuny.edu
Thomas Caswell is with National Synchrotron Light Source II, Brookhaven National Lab
E-mail: tcaswell@bnl.gov
Manuscript received X XX, XXXX; revised X XX, XXXX.

## II. Related Work

This paper builds on how structure has traditionally been discussed in visualization and mathematics and encapsulated in visualization library design to propose a uniform interface for encoding structure that supports a broader variety of fields and more rigorously define how connectivity is preserved. Generally, preserving structure means that a visualization is expected to preserve the $\texttt{field}$ properties and $\texttt{topology}$ of the corresponding dataset:

**field** [1] is a set of values of the same type, e.g. one column of a table or the pixels of an image

**topology** is the connectivity and relative positioning of elements in a dataset [4].

The conditions under which $\texttt{data} \rightarrow \texttt{graphic}$ is structure preserving is discussed extensively in the visualization literature, codified by Bertin[5] and extended to tool design by Mackinlay[6], and a set of conditions under which the $\texttt{graphic} \rightarrow \texttt{data}$ mapping is structure preserving is presented in Kindlemann and Scheidegger's algebraic visualization design (AVD) framework [7]. Encapsulating the AVD conditions, we present a uniform abstract data representation layer in subsection III-B for ensuring that the visualization should not change if the data representation (i.e. the data container) changes, define the conditions under which data is mapped unambiguously to visual encodings [8] in subsection III-C, and provide a methodology for verifying that changes in data should correspond to changes in the visualization in subsubsection IV-B2 that does not necessarily require that the changes be perceptually significant. Furthermore, our model generalizes the AVD notion of equivariance by allowing non-group structures, explicitly incorporating topology and by providing a framework for translating the theoretical ideas into buildable components in section V.

### A. Fields

Data is often described by its mathematical structure, for example the Steven's measurement scales define nominal, ordinal, interval, and ratio data by the allowed operations on each [9] and other researchers have since expanded the scales to encapsulate more types of structure [10], [11].

Loosely, the scales classify data as a set of values and the allowed transformations on that set, which can be operations, relations, or generalized as actions:

**Definition II.1.** [12] An **action** of $G = (G, \circ, e)$ on $X$ is a function $act : G \times X \rightarrow X$. An action has the properties of identity $act(e, x) = x$ for all $x \in X$ and associativity $act(g, act(f, x)) = act(f \circ g, x)$ for $f, g \in G$.

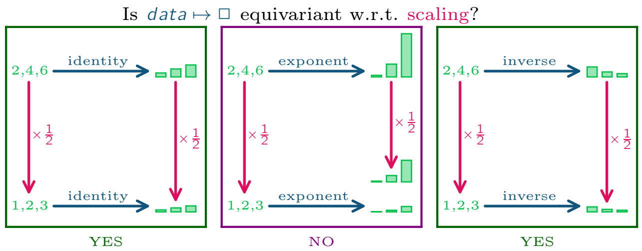



Fig. 1. Encoding data as the bar height using an exponential transform is not equivariant because encoding the data and then scaling the bar heights yields a much taller graph then scaling the data and then encoding those heights using the same exponential transform function.

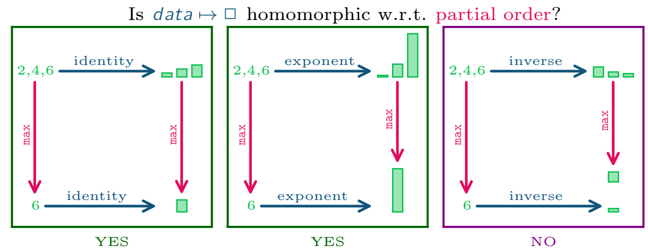



Fig. 2. Encoding data as bar height using an inverse transform is not homomorphic because the largest number is mapped to the smallest bar while the max function returns the largest bar.

Elements of X can be from one data field or all of them or some subset; similarly the actions act on the elements of X and each action can be a composition of actions. This means actions can be used when discussing various measures of structure preservation. For example, *equivariant* functions preserve structure under transformations to data or visualization and has been proposed by Kindlemann and Scheidegger[7] and *homomorphic* maps preserve relations between data elements was preserved as proposed by Mackinlay[6].

Specifically, Steven's conceptualizes the structure on values as actions on groups [2]. A function that preserves structure when the input or output is changed by a group action is called *equivariant*.

Given a group G that acts on both the input X and the output Y of a function $f : X \to Y$

**Definition II.2.** A function f is **equivariant** when $f(act(g, x)) = act(g, f(x))$ for all g in G and for all x in X [13]

which means that a visualization is structure preserving when there exist compatible group actions on the data and visualization, as discussed by Kindlemann and Scheidegger[7]. As illustrated in the commutative diagram in Figure 1, what this means is that the visual representation is consistent whether the data is scaled and then mapped to a graphic or whether the data is mapped to a graphic that is then modified in a compatible way.

Although the Steven's scales were conceptualized as having group structure, the ordinal scale has a monoidal structure because partial orders ($\geqslant, \leqslant$) are not invertible. This means *equivariance* cannot be used to test for structure preservation. Instead *homomorphism* can be used because it imposes fewer constraints on the underlying mathematical structure of the data.

Given the function $f : X \to Y$, with operators $(X, \circ)$ and $(Y, *)$

**Definition II.3.** A function f is **homomorphic** when $f(x_1 \circ x_2) = f(x_1) * f(x_2)$ and preserves identities $f(I_x) = I_y$ all $x, y \in X$ [12]

[2]A *group* is a set with an associative binary operator. This operation must have an identity element and be closed, associative, and invertible

which means that the operators $\circ$ and $*$ are compatible. In Figure 2, the $\geqslant$ operator is defined as the compatible closed functions `max` and the inverse transform is not homomorphic because it does not encode the maximum data value as the maximum bar value.

As shown in **??** and **??**, a function can be homomorphic but not equivariant, such as an exponential encoding, or equivariant but not homomorphic, such as the inverse encoding. A function can also be homomorphic (or equivariant) with respect to one action but not with respect to another. The encoding transforms in visualization tools are expected to preserve the structure of whatever input they receive; therefore a methodology for codifying arbitrary structure is presented in subsubsection III-A2 and subsubsection IV-B1 presents a generalization of equivariance and homomorphism for evaluating structure preservation.

## *B. Topology*

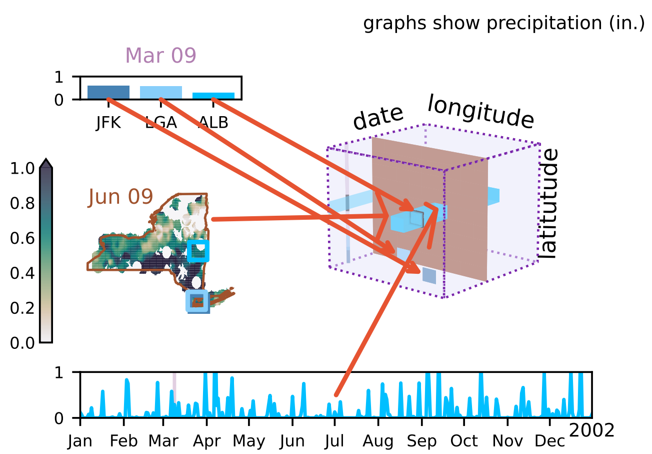



Fig. 3. This weather station data has multiple embedded continuities - points at each time and position, timeseries at each position, and maps at each time. The corresponding visualizations - bar chart, timeseries, and map - each preserve the continuity of the subset of the data they visualize by not introducing or leaving out values and preserving the relative positioning of continuous values.

Visual algorithms assume the topology of their input data, as described in taxonomies of visualization algorithms Chi[14] and by Troy and Möller [15], but generally do not verify that input structure. For example, a `line` algorithm often does not have a way to query whether a list of (x,y) coordinates is the distinct rows, the time series, or the list of stations in Figure 3. While plotting the time series as a continuous line would be correct, it would be incorrect for a visualization to indicate that the distinct rows or stations are connected in a 1D continuous manner because it introduces ambiguity over which part of the line maps back to the data. A map that by definition has continuous maps between the input and output spaces, such as data and graphics, is called a *homeomorphism*[16]:

**Definition II.4.** A function $f$ is a $\mathtt{homeomorphism}$ if it is bijective, continuous, and has a continuous inverse function $f^{-1}$.

The bar plot, line plot, and heatmap in Figure 3 have a homeomorphic relationship to the 0D (•) points, 1D (–) linear, and 2D (■) surface continuities embedded in the continuous 3 dimensional surface encapsulating time and position because each point of the visualization maps back into a point in its corresponding indexing space in the cube. Using homeomorphism to test whether continuity is preserved formalizes Bertin's codification of how the topology of observations matches the class of representation (i.e. point , line, area) [5] and Wilkinson's assertion that connectivity must be preserved [4].

To encode topology and field structure in a way that is both uniform and generalizable, we extend Butler's work on using a mathematical structure called fiber bundles as an abstract data representation in visualization [17], [18]. Using this topological model of indexing, semantic indexing as described by Munzner's key-value model of data structure [19] act as different ways of partitioning the underlying data continuity. For example, the data cube in Figure 3 could be subset into sets of timeseries where the key would be $\mathtt{station}$, or subset into maps where the key would be $\mathtt{date}$, or subset into station records where the keys are $(\mathtt{date}, \mathtt{latitude}, \mathtt{longitude})$. Using a topological model rather than semantic indexing also makes clearer when different labeling schemes refer to the same point, for example how 0-360 and 180E-180W are two ways of labeling longitude or how (date, lat, lon) and (date, station) refer to the same point. We sketch out fiber bundles in subsection III-A, but Butler provides a thorough introduction to bundles for visualization practitioners.

### C. Structure Preservation In Software

Visualization libraries are in part measured by how expressive the components of the library are, where expressiveness is a measure of which structure preserving mappings a tool can implement [20]. While some visualization tools aim to automate the pairing of data with structure preserving visual representations, such as Tableau[21]–[23], many visualization libraries leave that choice to the user. For example, connectivity assumptions tend to be embedded in each of the visual algorithms of 'building block' libraries, a term used by Wongsuphasawat [24], [25] to describe libraries that provide modular components for building elements of a visualization, such as functions for making boxes or translating data values to colors. In building block libraries such as Matplotlib[26] and D3[27] assumptions about connectivity are embedded in the interfaces such that the API is inconsistent across plot types. For example in Matplotlib methods for updating data and parameters for controlling aesthetics differ between (1D) line based plotting methods and (0D) marker based methods. While VTK[28], [29] provides a language for expressing the topological properties of the data, and therefore can embed that information in its visual algorithms, VTK's charts API is similar to the continuity dependent APIs of other building block libraries.

Domain specific libraries are designed with the assumption of continuities that are common in the domain [30], and therefore can somewhat restrict their API to choices that are appropriate for that domain. For example, a tabular topological structure of discrete rows, as illustrated in Figure 3, is assumed by A Presentation Tool[20] and grammar of graphics[4] and the ggplot[31], vega[32], and altair[33] libraries built on these frameworks. Image libraries such as Napari[34] and ImageJ[35] and the ImagePlot[36] art plugin assume that the input is a 2D continuous image. Networking libraries such as gephi[37] and networkx[38] assume a graph-like structure. By assuming the structure of their data, these domain specific libraries can provide more cohesive interfaces because a limited set of visualization algorithms apply to their data and the visualizations these algorithms provide can be styled in a fixed number of ways.

We propose that the cohesion of domain specific library APIs is obtainable using the uniform data model described in subsection III-A while the expressivity of building block libraries can be preserved by defining explicit structure preserving constraints on the library components, as described in section IV. Because category theory constructions map cleanly to objects and functions, using category theory to express the structure and constraints can lead to more consistent software interfaces in visualization software libraries [39], [40]. A brief visualization oriented introduction to category theory is in Vickers et al [41], but they are applying category theory to semantic concerns about visualization design rather than library architecture.

## III. Uniform Abstraction for Data & Graphics

In this section, we propose a mathematical abstraction of the data input and pre-rendered graphic output. This mathematical abstraction provides a uniform highly generalizable method for describing topology and fields; expresses how to verify that data continuity is preserved on subset, distributed, and streaming data representations; and formalizes the expectation of a correspondence between data and visual elements. Using these abstractions allows us to embed information about structure in dataset types:

$$\mathtt{dataset} : \mathtt{topology} \rightarrow \mathtt{fields} \tag{1}$$

which can then be checked by visualization algorithms to ensure that the assumptions of the data match the assumptions of the algorithm. This typing system also extends to pre-rendered graphic output, allowing us to develop the structure preservation framework in section IV that ensures that output fields are equivalent to the input fields and that the topology of the output graphic is homeomorphic to the topology of the input.

### A. Abstract Data Representation

We model data using a mathematical representation of data that can encode topological properties, field types, and data values in a uniform manner using a structure from algebraic topology called a fiber bundle. We extend Butler's proposal of using bundles as an abstraction for visualization data[17], [18] by incorporating Spivak's methodology for encoding named data types from his fiber bundle representation of relational databases [42], [43]. We build on this work to describe how to encode the connectivity of the data as a topological base space modeling the data indexing space, encode the fields as a fiber space that acts as the data schema (domain), and express the mappings between these two spaces as section functions that encode datasets as mappings from indexing space to field space `dataset: topology → fields`.

**Definition III.1.** A **fiber bundle** $(E, K, \pi, F)$ is a structure with topological spaces $E, F, K$ and bundle projection map $\pi : E \to K$ [44].

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

A continuous surjective map $\boldsymbol{\pi}$ is a **bundle projection** map when

1) all fibers in the bundle are isomorphic. Since all fibers are isomorphic $F \cong F_k$ for all points $k \in K$, there is a uniquely determined fiber space $F$ given by the preimage of the projection $\pi$ at any point $k$ in the base space $K$: $F = \pi^{-1}(k)$.
2) each point $k$ in the base space $K$ has an open neighborhood $U_k$ such that the total space $E$ over the neighborhood is locally trivial.

**Local triviality** means $E|_U = U \times F$. In this paper we use $E|_U = \pi^{-1}(U)$ to denote the preimage of an openset[3], and a **local trivialization** is a specific choice of neighborhoods (described in subsubsection III-A1) and their preimages such that the fibers in each preimage are identical $F = F_k$ for all points $k \in U$. All fiber bundles can be decomposed into sets of local trivializations that are also bundles and we can specify a gluing scheme that reconstructs the fiber bundle from locally trivial pieces by specifying**transition maps** for all overlaps of the local trivializations; therefore, while the framework in this paper applies to all bundles, in this paper we assume that the bundles are trivial bundles $E = K \times F$ so that we can assign all fibers in a bundle the same type. For an example of trivial and non-trivial bundles, see section B.

**Definition III.2.** A **section** $\tau : K \to E$ over a fiber bundle is a smooth right inverse of $\pi(\tau(k)) = k$ for all $k \in K$

$$\begin{tikzcd} F \arrow[r, hook] & E \arrow[d, two heads, "\pi"', shift right] \\ & K \arrow[u, dashed, bend right, "\tau"'] \end{tikzcd} \tag{3}$$

[3]Open sets (open subsets) are a generalization of open intervals to n dimensional spaces. For example, an open ball is the set of points inside the ball and excludes points on the surface of the ball. [45], [46]

We propose that the total space of a bundle can encode the mathematical space in which a dataset is embedded, the base space can encode the topological properties of the dataset, the fiber space can encode the data types of the fields of the dataset, and that the datasets can be encoded as section functions from the continuity to the fiber space.

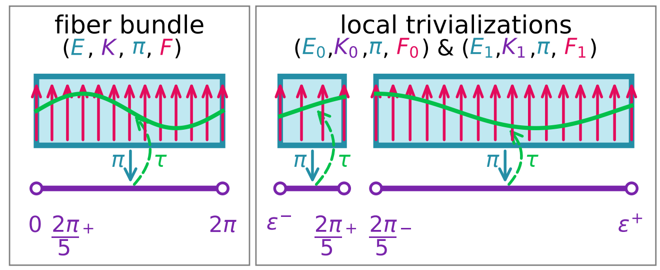



Fig. 4. The space of all data values encoded by this fiber bundle can be modeled as a rectangle total space. Each dataset in this data space lies along the interval $[0, 2\pi]$ base space. Each dataset has values along the $-1 \to 1$ interval fiber. One dataset embedded in this total space is the sin section over the bundle.

For example, the fiber bundle in Figure 4 encodes the space of all continuous functions that have a domain of $[0, 2\pi]$ and range $[0, 1]$. Using a fiber bundle abstraction encodes that the dataset has a 1D linear continuity as the base space Kis the interval $[0, 2\pi]$ and a field type that is a `float` in the range $[0, 1]$. Therefore the type signature of the datasets in this fiber bundle, which is called a section $\tau$, would be `dataset:` $[0, 2\pi] \to [0, 1]$. One such dataset (section) is the sin function, which as shown in Figure 4 is defined via a function $\tau$from a point in the base space to a corresponding point in the fiber. Evaluating the section function over the entire base space yields the sin curve that is composed of points intersecting each fiber over the corresponding point. The local trivializations shown in Figure 4 are one way of decomposing the total bundle and conversely the bundle can be constructed from the local trivializations $K = K_0 \oplus K_1$. As shown, the section sin spans the trivializations in the same manner that it spans the bundle; this is analogous to how a dataset may span multiple tables or be collected in one table. The trivializations are glued together into the bundle at the overlapping region $\frac{2\pi}{5}$ by defining the transition map $F_1 \to F_2$. Because the fibers in Figure 4 at $\frac{2\pi}{5}$ are aligned, the transition map is an identity map that take every value in $F_1$ and maps it to the same value in $F_2$ so that the sections, such as sin, remain continuous.

*1) Topological Structure: Base Space $K$:* We encode the topological structure of the data as the base space of a fiber bundle. Describing connectivity using the language of topology allows for describing individual elements in a way that holds true whether the data fits in memory, is distributed, or is streaming. This is because, informally, a topology $\mathcal{T}$ on the underlying data indexing space (which is a proxy for the continuity), is a partitioning of that space such that the partitions are of the same mathematical type as each other and the partitioned space. The partitions must also be composable in a continuity and property preserving way.

There are various equivalent definitions of topology, but here we use the neighborhood axiomatization because it is most analogous to the data access model of index (element) in subset (neighborhood) of all indices (mathematical space). Given a set $X$ and a function $\mathcal{N} : X \to 2^{2^X}$ that assigns to any $x \in X$ a non-empty collection of subsets $\mathcal{N}(x)$, where each element of $\mathcal{N}(x)$ is a *neighborhood of* $x$, then $X$ with $\mathcal{N}$ is a topological space and $\mathcal{N}$ is a neighborhood *topology* if for each $x$ in $X$: [47]

**Definition III.3.**
1) if $N$ is a neighborhood $N \in \mathcal{N}(x)$ of $x$ then $x \in N$
2) every superset of a neighborhood of $x$ is a neighborhood of $x$; therefore a union of a neighborhood and adjacent points in $X$ is also a neighborhood of $x$
3) the intersection of any two neighborhoods of $x$ is a neighborhood of $x$
4) any neighborhood $N$ of $x$ contains a neighborhood $M \subset N$ of $x$ such that $N$ is a neighborhood of each of the points in $M$

Therefore a neighborhood has to contain $x$ (1), can grow (2), or shrink (3), and every neighborhood also contains smaller neighborhoods of points adjacent to $x$ (4). For example, in the indexing cube in **??**, the brown surface and blue rectangle are both neighborhoods of the index for the measurement in Albany on June 09. The blue rectangle is also a neighborhood of the index for the measurement in Albany on March 09. The indexing cube is a neighborhood for both of these indices. While Definition III.3 applies broadly to topological spaces, in this paper we usually model the indexing space as CW-complexes. CW-complexes are a class of topological spaces built by gluing together n-dimensional balls (which include points, intervals, filled circles, filled spheres, etc.) using continuous attaching maps. Because the base space of a fiber bundle is a quotient topology[48], it divides the topological space into the largest number of open sets such that $\pi$ remains a continuous function. This means that the topology can be defined to have a resolution equal to the number of indices in a dataset such that the key (continuity)-value (data) pairing is always preserved.

Following from Spivak's categorical abstraction of a database [42], [43], we also propose that the structure of the data types be formally specified as the objects of a category.

**Definition III.4.** An **category** $\mathcal{C}$ consists of the following *data*:

1) a collection of *objects* $X \in \mathbf{ob}(\mathcal{C})$
2) for every pair of objects $X, Y \in \mathbf{ob}(\mathcal{C})$, a set of *morphisms* $X \xrightarrow{f} Y \in \mathrm{Hom}_{\mathcal{C}}(X, Y)$
3) for every object $X$, a distinct *identity morphism* $X \xrightarrow{id_x} X$ in $\mathrm{Hom}_{\mathcal{C}}(X, X)$
4) a *composition function* $f \in \mathrm{Hom}_{\mathcal{C}}(X, Y) \times g \in \mathrm{Hom}_{\mathcal{C}}(Y, Z) \to g \circ f \in \mathrm{Hom}_{\mathcal{C}}(X, Z)$

such that

1) *unitality:* for every morphism $X \xrightarrow{f} Y$, $f \circ id_x = f = id_y \circ f$
2) *associativity:* if any three morphisms $f, g, h$ are composable,

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[rrr, bend right, dashed, "h\circ(g\circ f)=(h\circ g)\circ f"'] & Y \arrow[r, "g"] & Z \arrow[r, "h"] & W \end{tikzcd}$$

then they are associative such that $h \circ (g \circ f) = (h \circ g) \circ f$ [16], [49]–[51].

The standard construction of a category from a topological space is that it has open set objects $U$ and inclusion morphisms $U_i \xrightarrow{\iota} U_j$ such that $U_i \subseteq U_j$[16]. The composability property expresses that inclusion is transitive, while associativity expresses that the inclusion functions can be curried in various equivalent groupings. By formally specifying the properties of the topological structure data types as $\mathcal{K}$, we can express that these are the properties that are required as part of the implementation of the data type objects.

*a) Joining indexing spaces:* $\oplus : \mathcal{K} \sqcup \mathcal{K} \to \mathcal{K}$*:* Joining two bundles on their base space, for example to glue the two trivializations in Figure 4 into a bundle, is the coproduct $K_a \sqcup_{K_c} K_b$ over a space $K_c$. A property of the coproduct is that the inclusion morphism must be commutative, which means that index spaces are combined on overlapping indices: For example, in Figure 5, the open interval $K_c$ is present in both $K_a$ and $K_c$; therefore $K_a$ and $K_b$ are glued together where they overlap at $K_c$. Two spaces have been combined correctly if every point in the overlap $k \in K^c$ is present in each of the spaces $k \in K^a$ and $k \in K^b$ such that it is present in the joined space $k \in K^a \sqcup_{K^c} K_b$. This simple test that the records are joined correctly is what allows us to reliably build larger datasets out of smaller ones, such as in the case of distributed and on demand datasets.

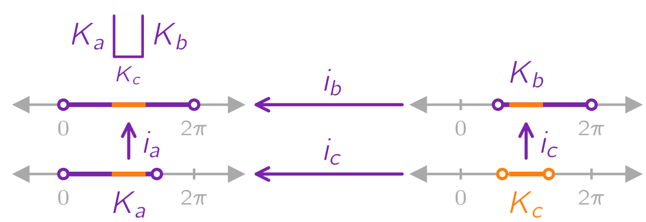



Fig. 5. Index spaces are combined via the *coproduct* $K_a \sqcup_{K_c} K_b$.

*2) Data Field Types: Fiber Space* $F$*:* As mentioned in subsection II-A, visualization researchers traditionally describe equivariance as the preservation of field structure, which is based on the field type. Spivak shows that data typing can be expressed in a categorical framework in his fiber bundle formulation of tables in relational databases [42], [43]. In this work, we adopt Spivak's definitions of *type specification*, *schema*, and *record* because that allows us to use a dimension agnostic named typing system for the fields of our dataset that is consistent with the abstraction we are using to express the continuity. Spivak introduces a *type specification* as a bundle map $\pi : \mathscr{U} \to \mathbf{DT}$. The base space $\mathbf{DT}$ is a set of data types

$\mathsf{T} \in \mathbf{DT}$ and the total space $\mathscr{U}$ is the disjoint union of the domains of each type

$$\mathscr{U} = \bigsqcup_{\mathsf{T} \in \mathbf{DT}} \pi^{-1}(\mathsf{T})$$

such that each element $x$ in the domain $\pi^{-1}(\mathsf{T})$ is one possible value of an object of type $\mathsf{T}$ [43]. For example, if $\mathsf{T} = \texttt{int}$, then the image $\pi^{-1}(\texttt{int}) = \mathbb{Z} \subset \mathscr{U}$ is the set of all integers and $x = 3 \in \mathbb{Z}$ is the value of one `int` object.

Since many fields can have the same datatype, Spivak formally defines a mapping from field name to field data type, akin to a database schema [52]. According to Spivak, a *schema* consists of a pair $(C, \sigma)$ where $C$ is the set of field names and $\sigma : C \to \mathbf{DT}$ is a function from field name to field data type[43]. The function $\sigma$ is composed with $\pi$ such that $\pi^{-1}(\sigma(C)) \subseteq \mathscr{U}$; this composition induces a domain bundle $\pi_\sigma : \mathscr{U}_\sigma \to C$ that associates a field name $c \in C$ with its corresponding domain $\pi_\sigma^{-1}(C) \subseteq \mathscr{U}_\sigma$.

**Definition III.5.** A **record** is a function $r : C \to \mathscr{U}_\sigma$ and the set of records on $\pi_\sigma$ is denoted $\Gamma^\pi(\sigma)$. Records must return an object of type $\sigma(C) \in \mathbf{DT}$ for each field $c \in C$.

Spivak then describes tables as sections $\tau : K \to \Gamma^\pi(\sigma)$ from an indexing space $K$ to the set of all possible records $\Gamma^\pi(\sigma)$ on the schema bundle, and his notion of a table generalizes to our notion of a data container.

To build on the rich typing system provided by Spivak, we define the fiber space $F$ to be the space of all possible data records

$$F \coloneqq \{r : C \to \mathscr{U}_\sigma \mid \pi_\sigma(r(C)) = C \text{ for all } C \in C\} \tag{4}$$

such that the preimage of a point is the corresponding data type domain $\pi^{-1}(k) = F_k = \mathscr{U}_{\sigma_k}$. Adopting Spivak's fiber bundle construction of types allows our model to reuse types so long as the field names are distinct and that field values can be accessed by field name, since those are sections on $\mathcal{U}_\sigma$. Furthermore, since domains $\mathscr{U}_\sigma$ of types are a mathematical space, multi-dimensional fields can be encoded in the same manner as single dimensional fields and fields can have different names but the same type.

As with the base space category $\mathcal{K}$, we propose a fiber category $\mathcal{F}$ to encapsulate the field types of the data. The fiber category has a single object $F$ of an arbitrary type and morphisms on the fiber object $\tilde{\phi} \in \mathrm{Hom}(F, F)$. We can also equip the category with any operators or relations that are part of he mathematical structure of the field type. For example we can equip the category with a comparison operator, which is part of the definition of the monoidal structure of a partially ordered ranking variable [53] or the group structure of Steven's ordinal measurement scale [9]–[11]. Steven's other scales are summarized in Table VI.

*a) Merging fields:* $\otimes : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$ *:* The fiber category $\mathcal{F}$ is also equipped with a bifunctor because it is a monoidal category and this functor provides a method for combining fiber types. The bifunctor allows $\otimes$ us to express fields that contain complexly typed values. For example, dates can be represented as three fields $F_{year} \times F_{month} \times F_{day}$ or a composite fiber field $F_{year} \times F_{month} \times F_{day} = F_{date}$. The $\otimes$ encapsulates both the sets of values associated with each fiber $\{y \in \mathbb{I} | 1992 \leqslant y \leqslant 2025\} \times \{m \in \mathbb{I} | 1 \leqslant m \leqslant 12\} \times \{d \in \mathbb{I} | 1 \leqslant d \leqslant 31\}$ and the composition function $\otimes : F_{year} \times F_{month} \times F_{day} \to F_{date}$ could include a constraint to only return dates that have the right number of days for each month. The bifunctor also composes the morphisms associated with each category into a morphism on the composite category $(\tilde{\phi}_{year}, \tilde{\phi}_{month}, \tilde{\phi}_{day}) = \tilde{\phi}_{date}$.

Combining fibers correctly can be verified by checking that when a fiber component $F^c$ is present in both $F^a$ and $F^b$, it is identical when projected out of either such that the *product* diagram commutes, means that when data with many fields is decomposed into its component fields, records maintain their integrity:

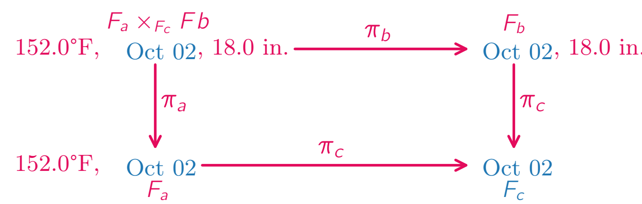



Fig. 6. Fields are combined via the *product* $F_a \times_{F_c} F_b$.

For example in Figure 6 the red (temperature, time, pressure) record separates into (temperature, time) and (pressure, time) records that share the same red time. Furthermore this time is the same whether it is obtained from the (temperature, time) or (pressure, time) record. Generally, the test for joining fields is that when a record is present in a fiber $r_c \in \pi_c(F_a), r_c \in \pi_c(F_b)$ then it is in the joint record $r_c \in \pi_c(F_a \times_{F_c} F_b)$ when is in the field $F_c$ being joined on. This simple test that fields are joined together correctly for the same record is what allows us to reliably combine multiple datasets together on shared properties, for example growing the weather station data in Figure 3 from a temporal to spatial dataset by adding the weather at each location at each time.

*3) Data: Section* $\tau$*:* We encode data as a section $\tau$ of a bundle because this allows us to incorporate the topology and field types in the data definition. We define section functions locally, meaning that the section is (piece-wise) continuous over a specific open subset $U$ of $K$

$$\Gamma(U, E\restriction_U) \coloneqq \left\{\tau : U \to E\restriction_U \;\middle|\; \pi(\tau(k)) = k \text{ for all } k \in U\right\} \tag{5}$$

such that each section function $\tau : k \mapsto r$ maps from each point $k \in U$ to a corresponding record in the fiber space $r \in F_k$ over that point. Bundles can have multiple sections, as denoted by $\Gamma(U, E \restriction U)$. We can therefore model data as structures that map from an index like point $k$ to a data record $r$, and encapsulate multiple datasets with the same fiber and base space as different sections of the same bundle.

When a bundle is trivial $E = K \times F$, we can defined a global sections $\tau : K \to F \in \Gamma(K, F)$ which we translate into a data signature of the form

$$\texttt{dataset} : \texttt{topology} \to \texttt{field} \tag{6}$$

where $\tau = \texttt{dataset}$, $K = \texttt{topology}$ and $F = \texttt{fields}$. When the bundle is non-trivial, we can use the fiber bundle property of local-triviality to define local sections $\tau \restriction_{U} \in \Gamma(U_k, E \restriction_{U_k})$. A local section is defined over an open neighborhood $k \in U \in K$, which is an open set that surrounds a point $k$. Most data sets can be encoded as a collection of local sections $\{\tau \restriction_{U_k} | k \in K\}$ and this encoding can be translated into a set of signatures

$$\{\texttt{data-subset} : \texttt{topology} \to \texttt{fields} \\ \text{s. t. } \texttt{data-subset} \subset \texttt{dataset}\} \tag{7}$$

The subsets of the fiber bundle and the transition maps between these subsets are encoded in an atlas[54] and the notion of an atlas can be incorporated into the data container, as discussed in subsection III-B.

*4) Example: Uniform Abstract Graphic Representation:* One of the reasons we use fiber bundles as an abstraction is that they are general enough that we can also encode the output of a visual algorithm as a bundle. We denote the output as a graphic, but the use of bundles allows us to generalize to output on any display space, such as a screen or 3D print.

$$D \hookrightarrow H \xrightarrow{\pi} S \tag{8}$$

The total space $H$ is an abstraction of an ideal (infinite resolution) space into which the graphic can be rendered. The base space $S$ is a parameterization of the display area, for example the inked bounding box in the cairo [55] rendering library. The fiber space $D$ is an abstraction of the renderer fields; for example a 2 dimension screen has pixels that can be parameterized $D = \{x, y\, z, r, g, b, a\}$. As with data, we model the graphic specifications as sections $\rho$ of the graphic bundle

$$\Gamma(W, H \restriction_{W}) := \\ \left\{\rho : W \to H \restriction_{W} \,\middle|\, \pi(\rho(s)) = s \text{ for all } s \in W\right\} \tag{9}$$

that map from a point in an openset in the graphic space $s \in W \subseteq S$ to a point in the graphic fiber $D$. The section evaluated on a single point $s$ returns a single graphic record, for example one pixel in an ideal resolution space. In our model, the unevaluated graphic section is passed from the visualization library component to the renderer to generate graphics.

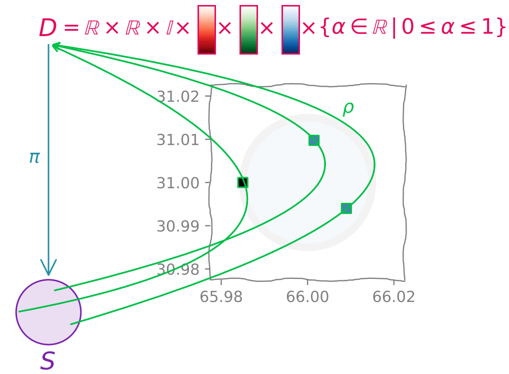



Fig. 7. The scatter marker is specified by the section $\rho$, which maps into the fiber $D$ to retrieve the values that compose into the pixel (approximated as a square) returned by the section function evaluated at each point $s$. The section evaluated over the entire space $\rho|_S$ returns the entire scatter mark, shown here in faded form to make it easier to see the individual pixels.

In Figure 7, the section function $\rho$ maps into the fiber for a simplified 2D RGB infinite resolution pre-render space and returns the $\{x, y, r, g, b\}$ values of a pixel in an infinite resolution space. In Figure 7 these pixels are approximated as the small colored boxes. Each pixel is the output of the $\rho(s)$ section that intersects the box. The set of all pixels returned by a section evaluated on a given visual base space $\rho|_S$ can yield a visual element, such as a marker, line, or piece of a glyph and in Figure 7 is a blue circle with a black edge. While Figure 7 illustrates a highly idealized space with no overlaps, overlaps can be managed via a fiber element $D_z$ for ordering. It is left to the renderer to choose how to blend $D_z$ and $D_a$ layers.

## *B. Abstract Data Containers*

While bundles provide a way to describe the structure of the data, sheaves are a mathematical way of describing the data container. Sheaves are an algebraic data structure that provides a way of abstractly discussing the bookkeeping that data containers must implement to keep track of the continuity of the data [54]. This abstraction facilitates representational invariance, as introduced by Kindlemann and Scheidegger[7], since the container level is uniformly specified as satisfying presheaf and sheaf constraints. When a data container satisfies these constraints, the subsets and whole data space have the same mathematical properties, e.g. morphisms, such that the framework in this paper applies whether the data is in memory, distributed, streaming, or on-demand.

We can mathematically encode that we expect data containers to preserve the underlying continuity of the indexing space and the mappings between indexing space and record space using a type of function called a functor. Functors are mappings between categories that preserve the domains, codomains, composition, and identities of the morphisms within the category[16].

**Definition III.6.** [56], [57] A **functor** is a map $F : \mathcal{C} \to \mathcal{D}$, which means it is a function between objects $F : \mathbf{ob}(\mathcal{C}) \mapsto \mathbf{ob}(\mathcal{D})$ and that for every morphism $f \in \mathrm{Hom}(C_1, C_2)$

there is a corresponding function $F : \mathrm{Hom}(C1, C2) \mapsto \mathrm{Hom}(F(C_1), F(C_2))$. A **functor** must satisfy the properties

- *identity*: $F(id_C(C)) = id_D(F(C))$
- *composition*: $F(g) \circ F(f) = F(g \circ f)$ for any composable morphisms $C_1 \xrightarrow{f} C_2$, $C_2 \xrightarrow{g} C_3$

$F(C) \in \mathbf{ob}(\mathcal{D})$ denotes the object to which an object $C$ is mapped, and $F(f) \in \mathrm{Hom}(F_(C_1), F_(C_2))$ denotes the morphism that $f$ is mapped to.

Modeling the data container as a functor allows us state that, just like a functor, the container is a map between index space objects and sets of data records that preserve morphisms between index space objects and data records.

$$\mathcal{O}_{K,E} : U \to \Gamma(U, E \restriction_U) \tag{10}$$

A common way of encapsulating a map from a topological space to a category of sets is as a presheaf

**Definition III.7.** A **presheaf** $F : \mathcal{C}^{op} \to \mathbf{Set}$ is a contravariant functor from an object in an arbitrary category to an object in the category $\mathbf{Set}$[44], [58].

A functor is contravariant when the morphisms between the input objects go in the opposite direction from the morphisms between the output objects. The presheaf is contravariant because the inclusion morphisms between input object $\iota : U_1 \to U_2$ are defined such that they correspond to the partial ordering $U_1 \subseteq U_2$, but the restriction morphisms $\iota^*$ between the sets of sections $\iota^* : \Gamma(U_2, E \restriction_{U_2}) \to \Gamma(U_1, E \restriction_{U_c 1})$ restricts the larger set to the smaller one such that all functions that are continuous over a space must be continuous over a subspace $\Gamma_2 \subseteq \Gamma_1$, where $\Gamma_i \coloneqq \Gamma(U_i, E \restriction_{U_i})$.

Data containers that implement managing subsets in a structure preserving way are satisfying the presheaf constraints that subsets of the indexing space $U_1$ are included in any index $U_2$ that is a superset $\iota$ and that data defined over an indexing space must exist over any indexes inside that space $\iota^*$. For example, lets define presheaves $\mathcal{O}_1, \mathcal{O}_2$. These are maps from intervals $U_1, U_2$ to a set of functions $\Gamma_1, \Gamma_2$ that are continuous over that interval:

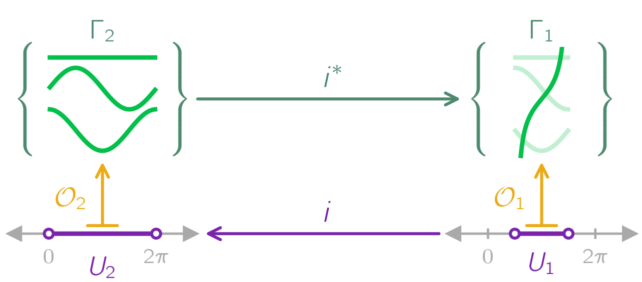



Fig. 8. Modeling this data container as a presheaf specifies that since cos, sin, $and c$ are continuous over $U_2$, they must be continuous over $U_1$ since $U_1$ is a subset of and therefore must be included in $U_2$. Because tan is only defined over $U_2$, it does not need to be included in the set $\Gamma_2$.

For example in Figure 8, since the $\mathtt{constant}, \mathtt{sin}, \mathtt{cos}$ functions are defined over the interval $[0, 2\pi]$, these functions must also be continuous over the sub-interval $\left(\frac{\pi}{2}, \frac{3\pi}{2}\right)$; therefore the sections in $\Gamma_2$ must also be included in the set of sections over the subspace $\Gamma_1$. One generalization of this constraint is that data structures that contain continuous functions must support interpolating them over arbitrarily small subspaces.

While presheaves preserve the rules for sets of sections, sheaves add on conditions for gluing individual sections over subspaces into cohesive sections over the whole space.These are the conditions that when satisfied ensure that a data container is managing distributed and streaming data in a structure preserving way.

**Definition III.8.** [44], [59] A **sheaf** is a presheaf that satisfies the following two axioms

- *locality* two sections in a sheaf are equal $\tau^a = \tau^b$ when they evaluate to the same values $\tau^a|_{U_i} = \tau^b|_{U_i}$ over the open cover $\bigcup_{i \in I} U_i \subset U$ (indexed by I).
- *gluing* the union of sections defined over subspaces $\tau^i \in \Gamma(U_i, E|_{openset_i})$ is equivalent to a section defined over the whole space $\tau|_{U_i} = \tau^i$ for all $i \in I$ if all pairs of sections agree on overlaps $\tau^i|_{U_i \cap U_j} = \tau^j|_{U_i \cap U_j}$

The gluing axiom says that a distributed representation of a dataset, which is a set of local sections, is equivalent to a section over the union of the opensets of the local sections. The gluing axiom can also be used to generate the gluing rules used to construct non-trivial bundles from the set of trivial local sections. The locality axiom asserts that the glued section function is equivalent to a function over the union if they evaluate to the same values.

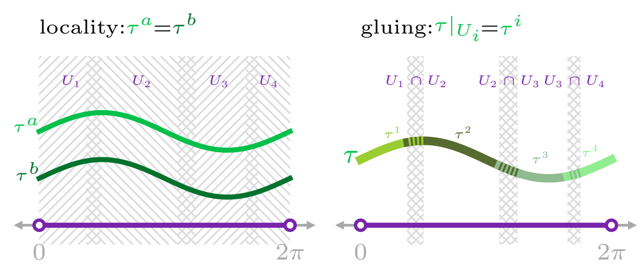



Fig. 9. A sheaf has the conditions that sections are equal when they match on all subsets (*locality*) and that the sections can be concatenated when they match on overlaps *gluing*.

For example, in Figure 9, the $\tau^a$ and $\tau^b$ sin sections are equal because they match *locally* on all subsets. This is true whether sin is defined over parts ($\sin|_{U_i}$) or the whole space. If sin is defined over parts, then those parts can be *glued* together. The concatenated sin is continuous because the pieces of the section outside the overlap are continuous with the pieces inside the overlap. The glued sin is also equal to the non-glued sin because they match on the opensets; therefore they are equivalent representations of the same section sin and so have the same mathematical properties.

While each section of a sheaf is evaluated over a point $\tau(k)$ such that it returns a single record, the sheaf model also provides an abstraction when neighboring information is required. The sheaf over a very small region surrounding a point $k$ is called a *stalk*[60]

$$\mathcal{O}_{K,E}\restriction_k := \lim_{U \ni k} \Gamma(U, E \restriction_U) \tag{11}$$

where the fiber is contained inside the stalk $F_k \subset \mathcal{O}_{K,E} \restriction_k$. The *germ* is the section evaluated at a point in the stalk $\tau(k) \in \mathcal{O}_{K,E} \restriction_k$ and is the data. Since the stalk includes the values near the limit of the point at $k$ the germ can be used to compute the mathematical derivative of the data for visualization tasks that require this information.

### C. Data Index and Graphic Index Correspondence

When data and graphic containers are modeled as sheaves and there is a continuous map between their base spaces, the properties of sheaves can be used to describe how visual elements correspond to distinct data elements, which is a necessary condition of a visualization being readable[8]. We propose that if a visualization is structure preserving, there exists a homeomorphic map $\xi$ between the graphic indexing space $S$ and the data indexing space $K$ that maps multiple graphic indexes to one data index such that every index in the graphic space can be mapped to an index in the data space. We define the mapping $\xi$ to be a surjective continuous map:

$$\xi : W \to U \tag{12}$$

between a graphic subspace $W \subseteq S$ and data subspace $U \subseteq K$. The set of points in graphic space that correspond to each point in data space is

$$\xi^{-1}(k) = \{s | \xi(s) = k \forall k \in K, s \in S\} \tag{13}$$

such that every point in a graphic space has a corresponding point in data space.

We construct the map as going from graphic to data because that encodes the notion that every visual element traces back to the data in some way, but there may be more data available than what is shown in the graphic. As exemplified in Figure 10, defining $\xi$ as a surjective map allows allows us to express visual representations of a single record that are the union of many primitives, each of which has a different base space $S_i$. These visual representations include multipart glyphs (e.g boxplots), combinations of plot types (e.g line with point markers), and the same point showing up in different continuities, such as $\tau(K_point)$ in Figure 10 and a station in Figure 3.

*1) Data and Graphic Correspondence:* Since we have defined a continuous function $\xi$ between two spaces $K, S$, we can construct functors that transport sheaves over each space to the other space[60]. This allows us to describe what data we expect is being visualized at each graphic index location and what graphic is generated for the data at each data index location. Transport functors compose the indexing map $\xi$ with the sheaf map to say that a record $\tau$ evaluated at a data index $k$ is the same record at all corresponding graphic indices $s$ and that a graphic specification $\rho$ over one point $s$ is the same specification at all points $s \in S$ that correspond to the same record index $k$.

*a) **Graphic Corresponding to Data**:* The pushforward (direct image) sheaf establishes which graphic generating function $\rho$ corresponds to a point $k \in dbase$ in the data base space.

**Definition III.9.** Given a sheaf $\mathcal{O}_{S,H}$ on $S$, the **pushforward** sheaf $\xi_* \mathcal{O}_{S,H}$ on $K$ is defined as

$$\xi_*(\mathcal{O}_{S,H})(U) = \mathcal{O}_{S,H}(\xi^{-1}(U)) \tag{14}$$

for all opensets $U \subset K$[60].

The pushforward sheaf returns the set of graphic sections over the data base space that corresponds to the graphic space $\xi^{-1}(U) = W$. The pushforward functor $\xi_*$ transports sheaves of sections on $W$ over $U$

$$\Gamma(U, \xi_* H \restriction_U) \ni \xi_* \rho : U \to \xi_* H \restriction_U \tag{15}$$

such that it provides a way to look up which graphic corresponds with a data index

$$\xi_* \rho(k) = \rho \restriction_{\xi^{-1}(k)} \tag{16}$$

such that $\xi_* \rho(k))(s) = \rho(s)$ for all $s \in \xi^{-1}(k)$. Therefore, the continuous map $\xi$ and transport functors $\xi^*, \xi_*$ allow us to express the correspondence between graphic section and data section. The parameterization of $\xi_* \rho$ in Figure 10 is intended as an approximation and is akin to declarative visualization specs such as vega [32] and svg [62]. These specs and $\xi_* \rho$ also provide a renderer independent way of describing the graphic and are therefore useful for standardizing internal representation of the graphic and serializing the graphic for portability.

*b) **Data Corresponding to Graphic**:* The pullback (inverse image) sheaf establishes which data record returned by $\tau$ corresponds to each point $s \in S$ in the graphic base space.

**Definition III.10.** [60] Given a sheaf $\mathcal{O}_{K,E}$ on $K$, the **pullback** sheaf $\xi^* \mathcal{O}_{K,E}$ on $S$ is defined as the sheaf associated to the presheaf $\xi^*(\mathcal{O}_{K,E})(W) = \mathcal{O}_{K,E}(\xi(W))$ for $\xi(W) \in K$.

The pullback sheaf returns the set of data sections over the graphic base space that corresponds to the graphic space $\xi(W) = U$. The pullback $\xi^*$ transports sheaves of sections on $U \subseteq K$ over $W \subseteq S$

$$\Gamma(W, \xi^* E \restriction_W) \ni \xi^* \tau : W \to \xi^* E \restriction_W \tag{17}$$

such that there is a way to then look up what data values correspond with a graphic index

$$\xi^* \tau(s) = \tau(\xi(s)) = \tau(k) \tag{18}$$

As $\xi$ is surjective, there are many points $s \in W \subseteq S$ in the graphic space that correspond to a single point $\xi(s) = k$. As illustared in Figure 10, $\xi^*$ is critical to interactive techniques such as brushing, linking, and tooltips[61] because they depend on being able to look up which data values go to which parts of the graphic.

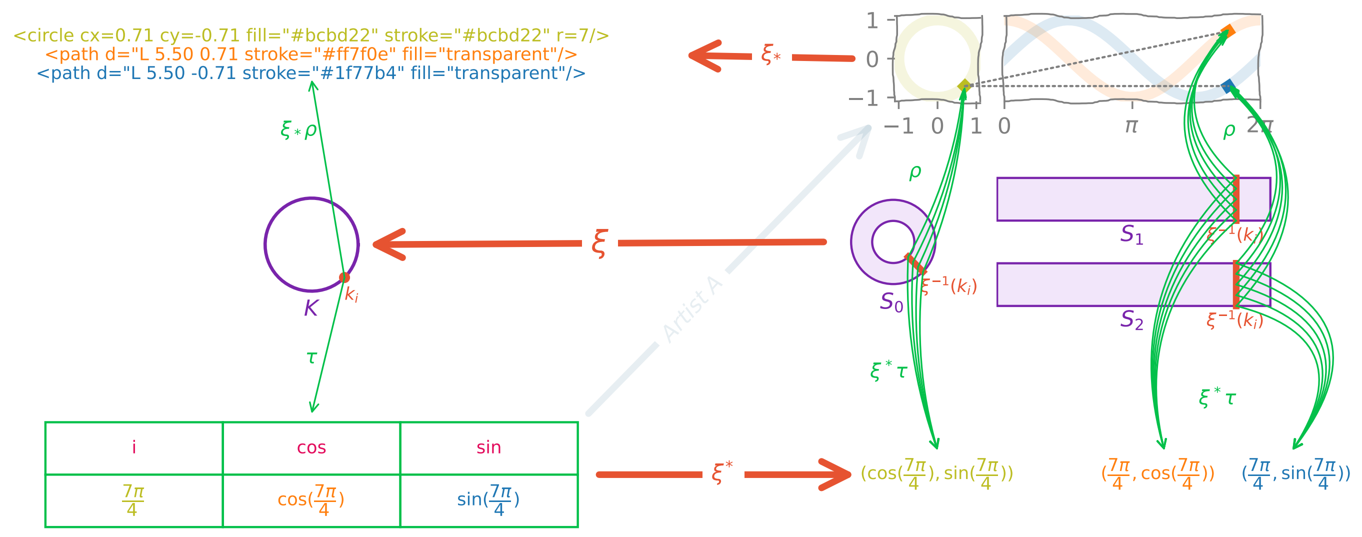



Fig. 10. The data $\tau$ consists of the $\mathtt{sin}$ and $\mathtt{cos}$ functions over a unit circle base space, which here are circle and two line plots generated from the graphic specification $\rho$The index map $\xi$ keeps track of which part of the circle, sin, and cosine plots correspond to which point on the unit circle. For example the orange bands on $S_0, S_1, S_2$ map to the orange point $k_i$. The pushforward $\xi_*$ matches each point in the data space to the specification of the graphic at that point, here illustared as an svg-like specification. The pullback $\xi^*$ matches each point in the graphic space to the data over that point, and as shown often the same data point maps to multiple graphic spaces (and their corresponding pixels).

*2) Example: Graphic and Data:* As illustrated in Figure 10, modeling the data and graphic containers as sheaves and constructing structure preserving maps between the indicies provides a method of precisely describing the which data values belong with which graphical representations. The $(k_i, S_i)$ pairing expressed in Equation 12 establishes that there is a correspondence between data at an index $k_i$ and graphics generated at $S_i$. In Figure 10, one example of this correspondance is the the orange bands on $S_0, S_1, S_2$ that correspond to the orange point $k_i$. Because of this correspondance and the properties of sheaves, we can construct a graphic specifications for each data index $\xi_*\rho$; in Figure 10 $\xi_*\rho(K_i)$ a psuedo-svg spec describing the piece of the circle, sin, and cosine curves generated for the data at the point $k$We can also retrieve the data $\xi^*\tau_{\xi^{-1}(k)}$ that is being visualized by each specific part of the graphic, which in Figure 10 is the section $\tau(k_i)$.

## IV. Codifying Structure Preservation

In this work we propose that visualization libraries are implementing transformations from data sheaf to graphic sheaf. We call these subset of functions the artist:

$$\mathcal{A} : \Gamma(K, E) \rightarrow \Gamma(S, H) \tag{19}$$

The artists can be constructed as morphisms of sheaves over the same base spaces through the application of pushforward and pullback functors; therefore they are natural transformations.

**Definition IV.1.** Given two functors $F, G$ with the same domain $\mathcal{C}$ and codomain $\mathcal{D}$, a **natural transformation** $\alpha : F \Rightarrow G$ is a map

- *data*: morphism $F(c) \xrightarrow{\alpha_c} G(c)$ for each object $c \in \mathcal{C}$
- *property* when $f : c_1 \to c_2$ is a morphism in $\mathcal{C}$, the components of the natural transform commute $G(f) \circ \alpha_{c_1} = \alpha_{c_2} \circ F(f)$

such that $\alpha = (\alpha_c)_{c\in\mathcal{C}}$ is the set of all natural transformation components $\alpha_c$.[63]

This means that natural transforms are maps of functors that take the same input object and return objects in the same category[64]. As illustrated in Equation 57, the sheaf functors

$$\Gamma(K, E) \xleftarrow{\mathcal{O}_{K,E}} K \xrightarrow{\xi_*\mathcal{O}_{S,H}} \Gamma(K, \xi_*H) \tag{20}$$

take as input an openset object $U$ or $W$ and return sets of data and graphic sections that are objects in $\mathbf{Set}$. As a map between these sheaf functors, the artist has to preserve the $\iota, \iota^*$ morphisms of the presheaf functor, described in ?? and ??, such that the following diagram commutes: this needs human words - subsets of functions of the same type map to subsets of visualizations of the same type

$$\begin{tikzcd} K_1 & \Gamma(K_1, E) \arrow[r, "A_{K_1}"] \arrow[d, "\iota^*"] & \Gamma(K_1, \xi_*H) \arrow[d, "\iota^*"] \\ K_2 \arrow[u, hook, "\iota"] & \Gamma(K_2, E) \arrow[r, "A_{K_2}"] & \Gamma(K_2, \xi_*H) \end{tikzcd} \tag{21}$$

The diagram in Equation 21 shows that restricting a set of outputs of an artist to a set of graphic sections over a subspace is equivalent to restricting the inputs to data sections over the same subspace. Because the artist is a functor of sheaves, the artist is expected to translate the data continuity to graphic continuity such that the connectivity of subsets is preserved. This bookkeeping is necessary for any visualization technique that selectively acts on different pieces of a data set; for example streaming visualizations [65] and panning and zooming [66]

The output of an artist $A$ is a restricted subset of graphic sections

$$\mathrm{Im}_A(S, H) := \{\rho \mid \exists \tau \in \Gamma(K, E) \text{ s.t. } A(\tau) = \rho,\ \xi(S) = K\} \tag{22}$$

that are, by definition, only reachable through a structure preserving artist, which we describe in subsubsection IV-B2. We define this subset because the space of all sections $\Gamma(W, H \restriction_U)$ includes sections that may not be structure preserving. For example, a section may go from every point in the graphic space to the same single point in the graphic fiber $\rho(s_i) = d\ \forall s \in S$ such that the visual output is a single inked pixel on a screen.

### A. Homeomorphism

As mentioned in subsection II-B, preserving the topology of a visualization means that each discrete piece of differentiable visual information corresponds to a distance element of the dataset[8] in a way where the organization of elements is preserved. A generalization of this condition is the idea that the graphic space can be collapsed into the data indexing space, which means that the data base space is a deformation retraction of the graphic base space[67]. By defining the indexing look up function $\xi$, introduced in subsection III-B, to be

$$\xi : K \times I \to K s.t. \xi(k) = k \forall k \in S \tag{23}$$

we can assert that the data space $K$acts as an indexing space into $S$such that knowing the location on space yields the location on the other and any point in either base space or graphic space has a corresponding point in the other space.

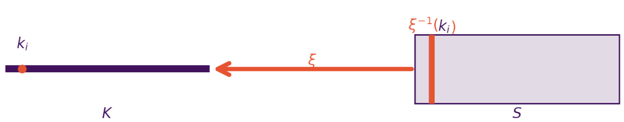



Fig. 11. The graphic base space $S$ is collapsible to the line $K$ such that every band $(k_i, [0, 1])$ on $S$ maps to corresponding point $k_i \in K$. The band $[0, 1]$ determines the thickness of a rendered line for a given point $k_i$ by specifying how pixels corresponding to that point are colored.

For example, as shown in Figure 11, a line is 1D but is a 2D glyph on a screen; therefore the graphic space $S$ is constructed by multiplying the base space $K$ with an interval $[0, 1]$. Because $S$ is collapsible into $K$, every band $(k_i, [0, 1])$ corresponds to a point in the base space $k_i \in K$. The first coordinate $\alpha = k_i$ provides a lookup to retrieve the associated visual variables. The second coordinate, which is a point in the interval $\beta = [0, 1]$. Together they are a point $s = (\alpha, \beta) \in gbase$ in the graphic base space. This point $s$ is the input into the graphic section $\rho(s)$ that is used to determine which pixels are colored, which in turn determines the thickness, texture, and color of the line.

### B. Equivariance

As introduced in subsection II-A, data and the corresponding visual encoding are expected to have compatible structure. This structure can be formally expressed as actions $\phi \in \Phi$ on the sheaf $\mathcal{O}_{K,E}$. We generalize from binary operations to a family of actions because that allows for expanding the set of allowable transformations on the data beyond a single operator. We describe the changes on the graphic side as changes in measurements $M$ which are scaler or vector components of the rendered graphic that can be quantified, such as the color, position, shape, texture, or rotation angle of the graphic. The visual variables [68] are a subset of measurable components. For example, a measurement of a scatter marker could be its color (e.g. red) or its x position (e.g. 5).

*1) Mathematical Structure of Data:* something something rotation etc We separate data transformations into two components, transformations on the base space $(\hat{\phi}, \hat{\phi}^*)$ and transformations on the fiber space $\tilde{\phi}$.

$$\begin{array}{ccc}
\Gamma(U, E \restriction_U) \xrightarrow{\hat{\phi}^*} \Gamma(U', \hat{\phi}^* E \restriction_{U'}) & \quad & \Gamma(U', \hat{\phi}^* E \restriction_{U'}) \\
\uparrow \qquad\qquad\qquad\qquad \uparrow & & \big\downarrow \tilde{\phi} \\
U \xleftarrow{\hat{\phi}} U' & & \Gamma(U', \hat{\phi}^* E \restriction_{U'})
\end{array} \tag{24}$$

The base space transformation transforms one openset object $U'$ to another object $U$, and the pullback functor transports the entire set of sections $\Gamma(U, E \restriction_U)$ over the new base space $\Gamma(U', \hat{\phi}^* E \restriction_{U'})$. The fiber transformation transforms a single section $\hat{\phi}^*\tau$ to a different section $\hat{\phi}^*\tau$.

*a)* ***Topological structure****:* The base space transformation is a point wise continuous map from one open set to another open set in the same base space

$$\hat{\phi} : k' \mapsto k \tag{25}$$

such that $U, U' \subseteq K$. This means $U$ and $U'$ are of the same topology type. To correctly align the sections with the remapped base space, there is a a corresponding section pullback function

$$\hat{\phi}^*\tau \restriction_{U'} : \tau \restriction_{U'} \mapsto \tau \restriction_{U' \circ \hat{\phi}} \tag{26}$$

such that $\tau|_U = \hat{\phi}^*\tau|_{U'}$ because $\tau|_U = \tau|_{\hat{\phi}(U')}$. This means that the base space transformation $\hat{\phi}(k') = \hat{\phi}(k)$ such that

$$\tau(K) = \hat{\phi}^*\tau(k') = \tau(\hat{\phi}(k')) \tag{27}$$

which means that the index of the record changes from $k$ to $k'$ but the values in the record are unmodified.

*b)* ***Records****:* As introduced in Equation 24, the fiber transformation $\tilde{\phi}$ is a change in section

$$\tilde{\phi} : \hat{\phi}^*\tau \restriction_{U'} \mapsto \hat{\phi}^*\tau' \restriction_U \tag{28}$$

where $\tau, \tau' \in \Gamma(U', \hat{\phi}^*E \restriction_{U'})$. Since $\tilde{\phi}$ maps from one continuous function to another, it must itself be continuous such that

$$\lim_{x \to k'} \tilde{\phi}(\hat{\phi}^*\tau(x)) = \tilde{\phi}(\hat{\phi}^*\tau(k')) \tag{29}$$

As mentioned in subsubsection III-A2, $\tilde{\phi}$ is also a morphism on the fiber category $\tilde{\phi} \in \mathrm{Hom}(\hat{\phi}^*F \restriction_{k'}, \hat{\phi}^*F \restriction_{k'})$ restricted to a point $k' \in U'$. This means $\tilde{\phi}$ has to satisfy the properties of a morphism (Definition III.4)

- *closed*: $\tilde{\phi}(\hat{\phi}^*\tau(k')) \in F$
- *unitality*: $\tilde{\phi}(\mathrm{id}_F(\hat{\phi}^*\tau(k'))) = \mathrm{id}_F(\tilde{\phi}(\hat{\phi}^*\tau(k')))$
- *composition and associativity*: $\tilde{\phi}(\tilde{\phi}(\hat{\phi}^*\tau(k'))) = (\tilde{\phi} \circ \tilde{\phi})(\hat{\phi}^*\tau(k'))$

Additionally, $\tilde{\phi}$ must preserve any features of $F$, such as operators that are defined as part of the structure of $F$. Examples of testing that $\tilde{\phi}$ preserves the operations, and therefore structure, of the Steven's measurement scales are shown in Table VI. We do not provide a general rule here because these constraints are defined with respect to how specific properties of the mathematical structure of individual fields $F$ are expected to be preserved rather than as a general consequence of $\tilde{\phi}$ being a section map and morphism of the category.

*c) **Topological structure and records**:* We define a full data transformation as one that induces both a remapping of the index space and a change in the data values

$$\phi : \tau \restriction_U \mapsto \tau' \restriction_U \circ \hat{\phi} \tag{30}$$

which gives us an equation that can express transformations that have both a base space change and a fiber change.

The data transform $\phi$ is composable

$$\phi = (\hat{\phi}, \prod_{i=0}^{n} \tilde{\phi}_i) \tag{31}$$

if each (identical) component base space is transformed in the same way $\hat{\phi}$ and there exists functions $\phi_{a,b} : E_a \times E_b \rightarrow E_a \times E_b$, $\phi_a : E_a \rightarrow E_a$ and $\phi_b : E_b \rightarrow E_b$ such that $\pi_a \circ \phi_a = \phi_{a,b} \circ \pi_a$ and $\pi_b \circ \phi_b = \phi_{a,b} \circ \pi_b$ then $\phi_{a,b} = (\phi_a, \phi_b)$. This allows us to define a data transform where each fiber transform $\tilde{\phi}_i$ can be applied to a different fiber field $F_i$.

$\tau$ = data

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |

$\hat{\phi}_E^*\tau$ = data.T

| | |
|---|---|
| 0 | 3 |
| 1 | 4 |
| 2 | 5 |

$\tilde{\phi}_E\tau$ = data*2

| | | |
|---|---|---|
| 0 | 2 | 4 |
| 6 | 8 | 10 |

$\phi_E\tau$ = data.T*2

| | |
|---|---|
| 0 | 6 |
| 2 | 8 |
| 4 | 10 |

Fig. 12. Values in a data set can be transformed in three ways: $\hat{\phi}$-values can change position, .e.g transposed; $\tilde{\phi}$-values can change, e.g. doubled; $\phi$ - values can change position and value

Figure 12 provides an example of a transposition base space change $\hat{\phi}$, a scaling fiber space change $\tilde{\phi}$, and a composition of the two $\phi$ applied to each data point $x_k \in \texttt{data}$. In the transposition only case, the values in $\hat{\phi}^*\tau$ retain their neighbors from $\tau$ because $\phi$ does not change the continuity. Each value in $\hat{\phi}^*\tau$ is also the same as in $\tau$, just moved to the new position. In $\tilde{\phi}\tau$, each value is scaled by two but remains in the same location as in $\tau$. And in $\phi\tau$ each function is transposed such that it retains its neighbors and all values are scaled consistently.

*2) Equivariant Artist:* We formalize this structure preservation as equivariance, which is that for every morphism on the data $(\hat{\phi}_E, \tilde{\phi}_E)$ there is an equivalent morphism on the graphic $(\hat{\phi}_H, \tilde{\phi}_H)$ The artist is an equivariant map if the diagram commutes for all points $s' \in S'$

$$\begin{tikzcd}
\Gamma(K, E) \arrow[rrr, "A"] \arrow[d, "\hat{\phi}^*{}_E"'] & & & \mathrm{Im}_A(S, H) \arrow[d, dotted, "\hat{\phi}^*{}_H"] \\
\Gamma(K', \hat{\phi}^*{}_E E) \arrow[ddd, "\tilde{\phi}_E"'] & K & S \arrow[l, "\xi"'] & \mathrm{Im}_A(S', \hat{\phi}^*{}_H H) \arrow[ddd, dotted, "\tilde{\phi}_H"] \\
& K' \arrow[u, "\hat{\phi}_E"] & S' \arrow[l, "\xi"'] \arrow[u, dotted, "\hat{\phi}_H"'] & \\
& & & \\
\Gamma(K', E') \arrow[rrr, "A"] & & & \mathrm{Im}_A(S', H')
\end{tikzcd} \tag{32}$$

such that starting at an arbitrary data point $\tau(k)$ and transforming it into a different data point and then into a graphic

$$A(\tilde{\phi}_E(\tau(\hat{\phi}_E(\xi(s'))))) = \tilde{\phi}_H(A(\tau(\xi(\hat{\phi}_H(s')))))$$

is equivalent to transforming the original data point into a graphic and then transforming the graphic into another graphic. The function $\hat{\phi}_H$ induces a change in graphic generating function that matches the change in data. The graphic transformation $\hat{\phi}_H$ is difficult to define because by definition it acts on a single record, for example a pixel in an idealized 2D screen.

Instead, we define an output verification function $\delta$ that takes as input the section evaluated on all the graphic space associated with a point $\rho_{\xi^{-1}\restriction_k}$ and returns the corresponding measurable visual components $M_k$. formall define $M$ as a space of measurements

$$\delta : (\rho \circ \xi^{-1}) \mapsto (K \xrightarrow{\delta_\rho} M) \tag{33}$$

The measurable elements can only be computed over the entire preimage because these aspects, such as thickness or marker shape, refer to the entire visual element.

$$\begin{tikzcd}
\Gamma(K, E) \arrow[r, "A"] \arrow[d, "\eta"'] & \mathrm{Im}_A(S, H) \arrow[d, "\mathtt{render}"] \arrow[dl, dashed, "\delta"'] \\
\mathrm{Hom}(K, M) & \mathtt{visualization} \arrow[l, "\mathtt{measure}"]
\end{tikzcd} \tag{34}$$

The extraction function is equivalent to measuring components of the rendered image $\delta = \mathtt{measure} \circ \mathtt{render}$, which means an alternative way of implementing the function when $S$ is not accessible is by decomposing the output into its measurable components.

We also introduce a function $\eta$ that maps data to the measurement space directly

$$\eta : \tau \mapsto (K \xrightarrow{\eta_\tau} M) \tag{35}$$

such that $\eta_\tau(k)$ is the expected set of measurements $M_k$. The pair of verification functions $(\eta, \delta)$ can be used to test that the expected encoding $\eta_\tau$ of the data matches the actual encoding $\delta_\rho$

$$\eta(\tau)(k) = \delta(A(\tau))(k) = \delta(\rho \circ \xi^{-1})(k) = M_k \tag{36}$$

An artist is equivariant when changes to the input and output are equivariant. As introduced in Equation 25, the base space transformation $\hat{\phi}$ is invariant because $\tau \restriction_U = \tau \restriction_{\hat{\phi}(U')}$. This means that, for all points in the data $k \in K$, the measurement should not change if only the base space is transformed

$$\eta(\tau)(\hat{\phi}(k')) = \delta(A(\tau))(k) \tag{37}$$

On the other hand, a change in sections Equation 28 induces an equivalent change in measurements

$$\eta(\tilde{\phi}(\tau))(k) = \tilde{\phi}_M(\delta(A(\tau))(k)) \tag{38}$$

The change in measurements $\tilde{\phi}_M$ is defined by the developer as the symmetry between data and graphic that the artist is expected to preserve.

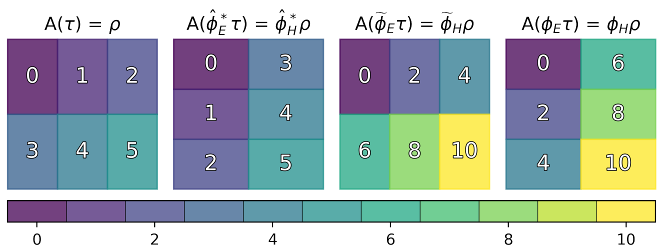



Fig. 13. This artist is equivariant because when the input data $\tau$ is transposed, $\hat{\phi}$, scaled $\tilde{\phi}$, and transposed and scaled $\phi$, the corresponding colored cells are transposed, scaled such that the color is moved two steps, and both transposed and scaled.

For example, in Figure 13, the measurable variable is color. This is a visual representation of the data shown in Figure 12, and as such the equivariant transformations are an equivalent transposition and scaling of the colors. This visualization is equivariant with respect to base space transformations, as defined in Equation 37, because the color values at the new position at the old position $measure'_k = M_k$. This visualization is also equivariant with respect to fiber wise transformations, as defined in Equation 38, because the colors are consistently scaled in the same was the data. For example, the values that have become 2 and 4 in the $\tilde{\phi}$ and $\phi$ panels are colored the same as the original 2 and 4 values in the first panel. The equivariance in this visualization is composable, as shown in the colors being both transposed and scaled correctly in the $\phi$ panel.

### C. Composing Artists

addition: intersections mapped same, multiplication: fibers mapped same large big data glued together correctly A common use of category theory in software engineering is the specification of modular components [39] such that we can build systems where the structure preserved by components is preserved in the composition of the components. This allows us to express that an artist that works on a dataset can be composed of artists that work on sub parts of that dataset.

*1) Addition:* We propose an addition operator that states that an artist that takes in a dataset can be constructed using artists that take as inputs subsets of the dataset

$$A_{a+b}(\Gamma(K^a \sqcup_{K^c} K^b, E)) \coloneqq A_a(\Gamma(K^a, E)) + A_b(\Gamma(K^b, E))$$

As introduce in $\mathrm{Equation}$ 19, the artist returns a function $\rho$. We assume that the output space is a trivial bundle, which means that $\rho \in \mathrm{Hom}(S, D)$ because the output specification is the same at each point $S$. This allows us to make use of the hom set adjoint propertyfind citation

$$\mathrm{Hom}(S^a + S^b, D) = \mathrm{Hom}(S^a, D) + \mathrm{Hom}(S^b, D)$$

to define an artist constructed via addition as consisting of two distinct graphic sections

$$\rho(s) \coloneqq \begin{cases} \rho^a(s) & s \in \xi^{-1}(K^a) \\ \rho^b(s) & s \in \xi^{-1}(K^b) \end{cases} \tag{39}$$

that are evaluated only if the input graphic point is an the graphic area that graphic section acts on.

One way to verify that these artists are composable is to check that the return the same graphic on points in the intersection $K^c$. Given $k_a \in K_c \subset K_a$ and $k_b \in K_c \subset K_b$, if $k_a = k_b$ then

$$\begin{aligned} &A_{a+b}(\tau^{a+b}(k_a)) \\ &= A_a(\tau^a(k_a)) = A_b(\tau^b(k_b)) \end{aligned} \tag{40}$$

for all $k_a, k_b \in K_a \bigsqcup_{K_c} K_b$

replace w/ a line plot w/markers One example of an artist that is a sum of artists is a sphere drawer that draws different quadrants of a sphere $A(\tau) = A_1(\tau_1) + A_2(\tau_2) + A_3(\tau_3)A_4(\tau_4)$. Given an input $k \in K_4$ in the 4th quadrant, then the graphic section that would be executed is $\rho_4$. If that point is also in the 3rd quadrant $k \in K_3$, then both artist outputs must return the same values $\rho_4(\xi^{-1}(k)) = \rho_3(\xi^{-1}(k))$.

*2) Multiplication:* fiber product vs cartesian product

In the trivial case where the base spaces are the same $K^a = K^b = K$, this is equivalent to adding more fields to a dataset.

$$A_{a\times b}(\Gamma(K, E^{a\times b})) \coloneqq A_a(\Gamma(K, E^a)) \times A_b(\Gamma(K, E^b))$$

which following from an adjoint property of homsets find citation and push this into a footnote or appendix maybe

$$\mathrm{Hom}(S, D) \times \mathrm{Hom}(S, D) = \mathrm{Hom}(S, D \times D) \tag{41}$$

which means that the artists on the subsets of fibers can be defined

$$\rho^{a\times b} = \{\rho^a(s), \rho^b(s)\}, s \in \xi^{-1}(K) \tag{42}$$

but that the signature of $\rho^{a\times b}$ would be $S \to D \times D$. Instead of having to special case the return type of artists that are compositions of multiple case, the hom adjoint find cite property

$$\mathrm{Hom}(S, D \times D) = \mathrm{Hom}(S + S, D)$$

means that multiplication can be considered as a special case of addition where $K^a = K^b$. While we discussed the trivial case in subsubsection IV-C1, there is no strict requirement that $F^a = F^b$.

One way to verify that these artists are composable is to check that they encode any shared fiber $F^c$ in the same way.

$$\begin{aligned}&\delta(A_{a\times b}(\tau^{a\times b}(k))) \restriction_{F^c}\\ &= \delta(A_a(\tau^a(k_a))) \restriction_{F^c} = \delta(A_b(\tau^b(k_b))) \restriction_{F^c}\end{aligned} \tag{43}$$

This expectation of using the same encoding for the same variable is a generalization of the concept of consistency checking of multiple view encodings discussed by Qu and Hullman [69]. This expectation can also be used to check that a multipart glyph is assembled correctly. For example, a box plot [70] typically consists of a rectangle, multiple lines, and scatter points; therefore a boxplot artist $A_{boxplot} = A_{rect} \times A_{errors} \times A_{line} \times A_{points}$ must be constructed such that all the sub artists draw a graphic at or around the same x value.

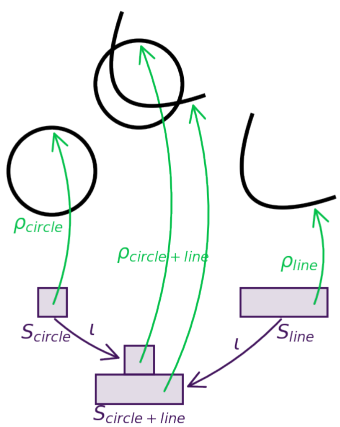



Fig. 14. The circle-line visual element can be constructed via $\rho_{circle}$ + $\rho_{line}$ functions that generate the circle and line elements respectively. This is equivalent to a $\rho_{circle+line}$ function that takes as input the combined base space $S_{circle} \sqcup S_{line} = S_{circle-line}$ and returns pixels in the circle-line element.

There is no way to visually determine whether a visual element is the output of a single artist or a multiplied or added collection of artists. The circle-line visual element in Figure 14 can be a visual representation of a highlighted point intersecting with a line plot with the same fields. The same element can also be encoding some fields of a section in the circle and other fields of that section in the lines. +*equive Although we have been discussing the trivial cases of adding observations or adding fields, this merging of artists in datasets can be generalized:

$$A(\Gamma(\bigsqcup_i K^i, \bigoplus_i E^i)) := \sum_i A_i(\Gamma(K^i, E^i)) \tag{44}$$

As shown in Equation 44, bundles over a union of base spaces can be joined as a product of the fibers. This allows us to consider all the data inputs in a complex visualization as a combined input, where some sections evaluate to null in fields for which there are no values for that point in the combined base space $k \in \sqcup_i K^i$ The combined construction of the data is a method for expressing what each data input has in common with another data input-for example the data for labeling tick marks or legends- and therefore which commonalities need to be preserved in the artists that act on these inputs.

explain why annotation is similar to brush/linking in operators section

### D. Animation and Interactivity

**pan, zoom, scroll** sheaf: locality + gluing Definition III.8
**selection and hover** pushforward Equation 16, pullback Equation 18
**brushing, linking, annotation** composition of artists Equation 44

Animation and interaction are a set of stills. Because the constraints are on the functions $A \circ \tau$, satisfying the constraints on each function means that the constraint is satisfied for all visualizations $\{A(\tau(k)) \mid k \in K\}$ that make up an animation or interaction.

## V. Constructing Structure Preserving Components

add a high level diagram Data-¿V-¿Screen add back in path of Q, use tikz backend to convert to pgf to then tweak We propose that one way of constructing artist functions is to separate generating a visualization into an encoding stage $\nu$ and a compositing stage $Q$. In the encoding stage $\nu$, a data bundle is treated as separable fields and each field is mapped to a measurable visual variable. In the encoding stage, many of the expected visual mappings $\eta$ can be implemented inside the library. Factoring out the encoding stage leaves the compositing stage $Q$ responsible for faithfully translating those measurable visual components into a visual element.

As mentioned in **??**, we construct the data base space as a deformation retraction of the graphic space. On simple way of doing so is to construct the graphic base space as a constant multiple of the base space such that

$$\underbrace{K \times [0,1]^n}_{S} \overset{\xi}{\longmapsto} K \tag{45}$$

where n is a thickening of the graphic base space S to account for the dimensionality of the output space

$$n = \begin{cases} \dim(S) - \dim(K) & \dim(K) < \dim(S) \\ 0 & \text{otherwise} \end{cases}$$

because otherwise the data dimensionality $K$ may be too small for a graphic representation. For example, as shown in Figure 11, a line is 1D but is a 2D glyph on a screen; therefore the graphic space $S$ is constructed by multiplying the base space $K$ with an interval $[0, 1]$.

### A. Measurable Visual Components

We encapsulate the space of measurable components reachable through the encoding stage $\nu$ as a visual fiber bundle $P \hookrightarrow V \xrightarrow{\pi} K$. The restricted fiber space $P$ of the bundle acts as the specification of the internal library representation of the measurable visual components. The space of visual sections $\Gamma(U, V\restriction_U) \coloneqq \{\mu : U \to V\restriction_U \mid \pi(\mu(k)) = k \text{ for all } k \in U\}$ return a visual encoding $\mu(k)$ corresponding to data record $k(k)$. Since the data bundle $\mathtt{dtotal}$ and visual bundle $V$ have the same continuity $\pi(\tau(k)) = \pi(\mu(k))$, they are considered structurally equivalent such that $E = V$. The distinguishing characteristic of $V$ is that it is part of the construction of the artist and therefore a part of the visualization library implementation. We propose that reusing the fibers $P$ across components facilitates standardizing internal types across the library and that this standardization improves maintainability (section C).

## B. Component Encoders

As introduced in subsection IV-B, there is a set $\eta$ of functions that map between data and corresponding visual encodings. We propose that for visualization library components to be structure preserving, they must implement a constrained subset of these encoding functions

$$\Gamma(K, E) \xrightarrow{\nu} \Gamma(K, V) \subset \Gamma(K, E) \xrightarrow{\eta} \mathrm{Hom}(K, M) \tag{46}$$

that preserve the categorical structure (operators and morphisms) of the fiber and the continuity of the data section. As mentioned in subsection V-A, the total visual space is restricted to the space of data types internal to the library $P \subset M$ and sections are subsets of homsets $\Gamma(K, V) \subset \mathrm{Hom}(K, M)$ because sections must be continuous.

The encoding functions $\nu$ are fiber wise transforms such that $\pi(E) = \pi(\nu(E))$. A consequence of this property is that $\nu$ can be constructed as a point wise transformation such that

$$\nu : F_k \to P_k \tag{47}$$

which means that means that a point in a single data fiber $r \in F_k$ can be mapped into a corresponding point in a visual fiber $V \in P_k$. This means that an encoding function $\nu$ can convert a single record independent of the whole dataset.

Since $E$ and $\nu\mathtt{total}$ are structurally identical, any $V$ can be redefined as $E$; therefore, as shown in Equation 48, any collection of $\nu$ functions can be composed such that they are equivalent to a $\nu$ that directly converts the input to the output.

$$\begin{tikzcd} F_k \arrow[r, "\nu"] \arrow[rr, bend right, dashed, "\nu''"'] & P_k \coloneqq F'_k \arrow[r, "\nu'"] & P'_k \end{tikzcd} \tag{48}$$

As with artists, $\nu$ are maps of sections such that the operators defined in subsection IV-C can also act on transformers $\nu$, meaning that encoders can be added $\nu_{a+b} = \nu_a + \nu_b$ and multiplied d $\nu_{a\times b} = \nu_a \nu_b$. Encoders designed to satisfy these composability constraints provide for a rich set of building blocks for implementing complex encoders.

*1) Encoder Verification:* A motivation for constructing an artist with an encoder stage $\nu$ is so that the conversion from data to measurable component can be tested separately from the assembly of components into a glyph.

$$\begin{tikzcd} F^a_k \times F^b_k \arrow[r, "\nu_{ab}"] \arrow[d, "\pi_a"'] \arrow[rr, bend left, "\eta_{ab}"] & P^a_k \times P^b_k \arrow[d, "\pi_a"] & M^{ab}_k \arrow[d, "M\restriction_a"] \\ F^a_k \arrow[r, dashed, "\nu_a"] & P^a_k \arrow[r, dotted, "\simeq"] & M^a_k \\ F^a_k \times F^c_k \arrow[r, "\nu_{ac}"] \arrow[u, "\pi_a"] \arrow[rr, bend right, "\eta_{ac}"'] & P^a_k \times P^c_k \arrow[u, "\pi_a"'] & M^{ac}_k \arrow[u, "M\restriction_a"'] \end{tikzcd} \tag{49}$$

As shown in Equation 49, an encoder is considered valid if there is an isomorphism between the actual outputted visual component and the expected measurable component encoding. An encoder is consistent if it encodes the same field in the same way even if coming from different data sources.

An encoding function $\nu$ is equivariant if the change in data, as defined in subsubsection IV-B1, and change in visual components are equivariant. Since $E$ and $V$ are over the same base space and are point wise, the base space change $\hat{\phi}_E$ applies to both sides of the equation

$$\nu(\tau_E(\hat{\phi}_K(k'))) = \mu(\hat{\phi}_K(k')) \tag{50}$$

and therefore there should not be a change in encoding. On the other hand, a change in the data values $\tilde{\phi}_E$ must have an equivalent change in visual components

$$\tilde{\phi}_V \nu(\tau(k)) = \nu(\tilde{\phi}_E(\tau(k))) \tag{51}$$

The change in visual components $\tilde{\phi}_V$ is dependent both on $\tilde{\phi}_E$ and the choice of visual encoding. As mentioned in subsection II-A, this is why Bertin and many others since have advocated choosing an encoding that has a structure that matches the data structure[5]. For example choosing a quantitative color map to encode quantitative data if the $\tilde{\phi}$ operation is scaling, as in Figure 13.

## C. Graphic Compositor

The compositor function $Q$ transforms the measurable components into properties of a visual element. The compositing function $Q$ transforms the sections of visual elements $\mu$ into sections of graphics $\rho$.

$$Q : \Gamma(K, V) \to \Gamma(S, H) \tag{52}$$

The compositing function is map from sheaves over $K$ to sheaves over $S$. This is because, as described in Figure 11, the graphic section must be evaluated on all points in the graphic space to generate the visual element corresponding to a data record at a single point $A(\tau(k)) = \rho(\xi^{-1}(k))$.

Since encoder functions are infinitely composable, as described in Equation 48, a new compositor function $Q$ can be constructed by pre=composing $\nu$ functions with the existing $Q$.

$$
\begin{tikzcd}
\Gamma(K, V) \arrow[r, "\nu"] \arrow[rr, dashed, bend right, "Q'"'] & \Gamma(K, V') \arrow[r, "Q"] & \Gamma(S, H)
\end{tikzcd}
\tag{53}
$$

The composition in Equation 53 means that different measurable components can yield the same visual elements. The operators defined in subsection IV-C can also act on compositors Q such that $Q_{a+b} = Q_a + Q_b$ and multiplied d $Q_{a\times b} = Q_a Q_b$.

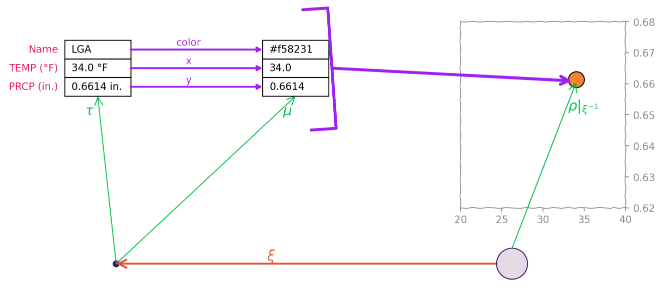



Fig. 15. This simple Q assembles a circular visual element that is the color specified in $\mu(k)$ and is at the intersection specified in $\mu(k)$ much better labeling, include semantic labeling, make everything bigger

As shown in Figure 15, a set of $\nu$ functions individually convert the values in the data record to visual components. Then the Q function combines these visual encodings to produce a graphic section $\rho$. When this section is evaluated on the graphic space associated with the data $\rho(\xi^{-1}(k))$, it produces a blue circular marker at the intersection of the x and y positions listed in $\mu$. The composition rule in Equation 53 means that developers can implement Q as drawing circles or can implement a Q that draws arbitrary shapes, and then provide different $\nu$ adapters, such as one that specifies that the shape is a circle.

*1) Compositor Verification:* An advantage of factoring out encoding and verification, as discussed in subsubsection V-B1, is that the responsibility of the compositor can be scoped to translating measurable components into visual elements.

$$
\begin{tikzcd}
\Gamma(K, V^a \times V^b) \arrow[r, "Q_{ab}"] \arrow[d, "\pi_a"'] & \mathrm{Im}_{\underset{ab}{A}}(S, H) \arrow[d, "M\restriction_a \circ \delta_{ab}"] \\
\Gamma(K, V^a) \arrow[r, dotted, "\simeq"] & \mathrm{Hom}(K, M^a) \\
\Gamma(K, V^a \times V^c) \arrow[r, "Q_{ac}"] \arrow[u, "\pi_a"] & \mathrm{Im}_{\underset{ac}{A}}(S, H) \arrow[u, "M\restriction_a \circ \delta_{ac}"']
\end{tikzcd}
\tag{54}
$$

As illustrated in Equation 54, a compositor is valid if there is an isomorphism between the actual outputted measured visual component and the expected measurable component that is the input. One way of verifying that a compositor is consistent is by verifying that it passes through one encoding even while changing others. For example, when $Q_{ab} = Q_{ac}$ then the output should differ in the same measurable components as $\mu_{ab}$ and $\mu_{ac}$.

A compositor function Q is equivariant if the renderer output changes in a way equivariant to the data transformation defined in subsubsection IV-B1. This means that a change in base space $\hat{\phi}_E$ should have an equivalent change in visual element base space. This means that there should be no change in visual measurement

$$\mu(\hat{\phi}_K(k')) = \delta(Q(\mu)(\hat{\phi}_K(\xi^{-1k}))) = M_k \tag{55}$$

As discussed in Figure 13, the change in base space may induce a change in locations of measurements relative to each other in the output; this can be verified via checking that all the measurements have not changed relative to the original positions $M_k = M_{k'}$ and through separate measurable variables that encode holistic data properties, such as orientation or origin.

The compositor function is also expected to be equivariant with respect to changes in data and measurable components

$$\tilde{\phi}_V(\mu(k)) = \tilde{\phi}_M(Q(\mu(k))) \tag{56}$$

which means that any change to a measurable component input must have a measurably equivalent change in the output. As illustrated in Figure 13, the compositor Q is expected to assemble the measurable components such that base space changes, for example transposition, are reflected in the output; faithfully pass through equivariant measurable components, such as scaled colors; and ensure that both types of transformations, here scaling and transposition, are present in the final glyph.

## D. Implementing the Artist

When a sheaf is equipped with transport functors, then the functions between sheaves over one space are isomorphic to functions between sheaves over the other space[60] such that the following diagram commutes

should either be oriented same as 55 and/or pushed back up to 3.3 as an intro to artist or squished a little.

$$
\begin{tikzcd}
\Gamma(U, E\restriction_U) \arrow[r, "\xi^*"] \arrow[d, "\mathrm{Hom}_{\mathcal{O}_K}"'] \arrow[dr, "\mathrm{Hom}_{\mathcal{O}_K \cdot \mathcal{O}_S}"] & \Gamma(W, \xi^* E\restriction_W) \arrow[d, "\mathrm{Hom}_{\mathcal{O}_S}"] \\
\Gamma(U, \xi_* H\restriction_U) & \Gamma(W, H\restriction_W) \arrow[l, "\xi_*"]
\end{tikzcd}
\tag{57}
$$

Since the artist is a family of functions in the homset between sheaves, the isomorphism allows for the specification of the transformation from data as combination of functions over different spaces such that the following diagram commutes:

$$\begin{tikzcd}
\Gamma(K, E) \arrow[r, "\nu^K"] \arrow[d, "\xi^*"'] \arrow[rr, bend left, "A^K"] \arrow[rrd, "A"] & \Gamma(K, V) \arrow[r, "Q^K"] \arrow[d, "\xi^*"] \arrow[rd, "Q"] & Im_A(K, \xi_* H) \\
\Gamma(S, \xi^* E) \arrow[r, "\nu^S"] \arrow[rr, bend right, "A^S"'] & \Gamma(S, \xi^* V) \arrow[r, "Q^S"] & Im_A(S, H) \arrow[u, "\xi_*"']
\end{tikzcd} \tag{58}$$

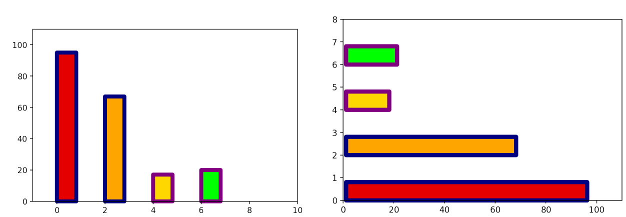
This means that an artist over data space $A_K : \tau \mapsto \xi_* \rho$, an artist over graphic space $vartists_S : \xi^* \tau \mapsto \rho$, and an artist $A : \tau \mapsto \rho$ are equivalent such that:

$$\begin{aligned}
&\tau(k) = \xi^* \tau(s) \\
\implies\ & A_K(\tau(k)) = A_S(\xi^* \tau(s)) = A(\tau(k)) \\
\implies\ & \xi_* \rho(s) = \rho(s)
\end{aligned}$$

when $\xi(s) = k$. This equivalence allows a developer to connect transformations over data space, denoted with a subset K, with transformations over graphic space S, using $\xi_*$ and $\xi^*$ adaptors.This allows developers to for example connect transformers that transform data on a line to a color in data space, but build a line compositing function that dynamically resamples what is on screen in graphic space.

## VI. Discussion: Feasibility as Design Spec

The framework specified in section IV and section V describes how to build structure preserving visualization components, but it is left to the library developer to follow these guidelines when building and reusing components. In this section, we introduce a toy example of building an artist out of the components introduced in section V to illustrate how components that adhere to these specifications are maintainable, extendible, scalable, and support concurrency.



(a) (b)

Specially, we introduce artists for building the graphical elements shown in  VI because it is a visualization type that allows us to demonstrate composability and multivariate data encoding. We build our visualization components by extending the Python visualization library Matplotlib's artist[4][26], [71] to show that components using this model can be incorporated into existing visualization libraries iteratively. While the architecture specified in section V can be implemented fully functionally, we make use of objects to keep track of parameters passed into artists. In this toy example, the small composable components allow for more easily verifying that each component does its transformation correctly before assembling them into larger systems.

[4]Matplotlib artists are our artist's namesake

### A. Bundle Inspired Data Containers

| fruit | calories | juice |
|---|---|---|
| apple | 95 | True |
| orange | 67 | True |
| lemon | 17 | False |
| lime | 20 | False |

We construct a toy dataset with a discrete K of 4 points and a fiber space of $F = \{apple, orange, lemon\} \times \mathbb{Z}^+ \times \{\texttt{True}, \texttt{False}\}$. We thinly wrap subsection VI-A in an object so that the common data interface function is that $\tau =$ `DataContainerObject.query`.

```python
class FruitFrameWrapper:
  def query(self, data_bounds, sampling_rate):
    # local sections are a list of
    # {field: local_batch_of_values}
    return local_sections
```

This interface provides a uniform way of accessing subsets of the data, which are local sections. The motivation for a common data interface is that it would allow the artist to talk to different common python data containers, such as numpy[72], pandas[73], xarray [74], and networkx[38]. Currently, data stored in these containers must be unpacked and converted into arrays and matrices in ways that either destroy or recreate the structure encoded in the container. For example a pandas data frame must be unpacked into its columns before it is sent into most artists and continuity is implicit in the columns being the same length rather than a tracked base space K. Because it is more efficient to work with the data in column order, we often project the fiber down into individual components. As shown in **??**, we can verify that this projection is correct by checking that the values at the index are the same regardless of the level of decomposition.

### B. Component Encoders

To encode the values in the dataset, we enforce equivariance by writing $\nu$ encoders that match the structure of the fields in the dataset. For example, the fruit column is a nominal measurement scale. Therefore we implement a position encoder that respects permutation $\hat{\phi}$ transformations. The most simple form of this $\nu$ is a python dictionary that returns an integer position, because Matplotlib's internal parameter space expects a numerical position type.

```python
def position_encoder(val):
  return {'apple': 0, 'orange': 2, 'lemon': 4, 'lime':
  ↪  6}[val]
```

As mentioned in Equation 48, the encoders can be composed up. For example, the compositor $\nu$ may need the position to be converted to screen coordinates. Here the screen coordinate $\nu$ is a method of a Matplotlib axes object; a Matplotlib axes is akin to a container artist that holds all information about the sub artists plotted within it.

```python
def composite_x_transform(ax, nu):
    return lambda x: ax.transData.transform(
            (position_encoder(x), 0))[0]
```

This encoder returns a function that is `transData.transform` $\nu_{\mathtt{transData}}$ composed with the position encoder $\nu_{\mathtt{position}}$ and takes as input a record to be encoded. As with the position encoder, the transData encoder respects permutation transforms because it returns reals; therefore the composite encoder respects permutation transforms. In this model, developers implement $\nu$ encoders that are explicit about which $\phi_V$ they support. Writing semantically correct encoders is also the responsibility of the developer and is not addressed in the model. For example `fruit_encoder = lamda x: {'apple': green, 'orange':'yellow', 'lemon':'red', 'lime':'orange'}` is a valid color encoding with respect to permutation, but none of those colors are intuitive to the data. It is therefore left to the user, or domain specific library developer, to choose $\nu$ encoders that are appropriate for their data.

### C. Graphic Compositors

After converting each record into an intermediate visual component $\mu$, the set of visual records is passed into $Q$. Here the $Q$ includes one last encoder, as illustrated in Equation 53, that assembles the independent visual components into a rectangle. This $\nu$ is inside the $Q$ to hide that library preferred format from the user. It is called `qhat` to indicate that this is the $A^K$ path in Equation 58. This means that the parameters are constructed in data space $K$ and this function returns a pushed forward $\xi_* \rho$.

```python
def qhat(position, width, length, floor, facecolor,
↪   edgecolor, linewidth, linestyle):
     box = box_nu(position, width, length, floor)
     def fake_draw(render,
     ↪   transform=mtransforms.IdentityTransform()):
         for (bx, fc, ec, lw, ls) in zip(box, facecolor,
         ↪   edgecolor, linewidth, linestyle):
             gc = render.new_gc()
             gc.set_foreground((ec.r, ec.g, ec.b, ec.a))
             gc.set_dashes(*ls)
             gc.set_linewidth(lw)
             render.draw_path(gc=gc, path=bx,
             ↪   transform=transform, rgbFace=(fc.r, fc.g,
             ↪   fc.b, fc.a))
     return fake_draw
```

The function `fake_draw` is the analog of $\xi_* \rho$. This function builds the rendering spec through the renderer API, and this curried function is returned. The transform here is required for the code to run, but is set to identity meaning that this function directly uses the output of the position encoders. The curried `fake_draw` $\approx \xi_* \rho$ is evaluated using a renderer object. In our model, as shown in Equation 34, the renderer is supposed to take $\rho$ as input such that $\mathtt{renderer}(\rho) = \mathtt{visualization}$, but here that would require an out of scope patching of the Matplotlib render objects.

One of the advantages of this model is that it allows for succinctly expressing the difference between two very similar visualizations, such as 16a and 16b. In this model, the horizontal bar is implemented as a composition of a $\nu$ that renames fields in $\mu_{\mathtt{barh}}$ and the $Q$ implementation for the horizontal bar.

```python
def qhat(length, width, position, floor, facecolor,
↪  edgecolor, linewidth, linestyle):
  return Bar.qhat(**BarH.bar_nu(length, width, position,
  ↪  floor, facecolor, edgecolor, linewidth, linestyle))
```

This composition is equivalent to $Q_{\mathtt{barh}} = Q_{\mathtt{bar}} \circ \nu_{\mathtt{vtoh}}$, which is an example of Equation 53. These functions can be further added together, as described in subsection IV-C to build more complex visualizations.

### D. Integrating Components into an Existing Library

The $\nu$ and $Q$ are wrapped in a container object that stores the $A = Q \circ \nu$ composition and a method for computing the $\mu$.

```python
class Bar:
  def compose_with_nu(self, pfield, ffield,
        nu, nu_inv:):
      # returns a new copy of the Bar artist
      # with the additional nu that converts
      # from a data (F) field value to a
      # visual (P) field value
      return new

  def nu(self, tau_local): #draw
     # uses the stored nus to convert data
     # stored nus have F->P field info
    return mus

  @staticmethod
  def qhat(position, width, length, floor, facecolor,
  ↪  edgecolor, linewidth, linestyle):

      return fake_draw
```

As shown in the `draw` method, generating a graphic section $\rho$ is implemented as the composition of `qhat` $\approx Q$ and `nu` $\approx \nu$ applied to a local section of the sheaf `self.section.query` $\approx \tau^i$ such `draw` $\approx Q \circ \nu \circ \tau = A \circ \tau$. The $\nu$ and $Q$ functions shown here are written such that they can generate a visual element given a local section $\tau\restriction_{K^i}$ which can be as little or large as needed. This flexibility is a prerequisite for building scalable and streaming visualizations that may not have access to all the data.

This artist is then passed along to a shim artist that makes it compatible with existing Matplotlib objects (section D). This shim object is hooked into the Matplotlib draw tree to produce the vertical bar chart in 16a. Using the Matplotlib artist framework means this new artist can be composed with existing artists, such as the ones that draw the axes and ticks. The example in this section is intentionally trivial to illustrate that the math to code translation is fairly straightforward and results in fairly self contained composable functions. A library applying these ideas, created by Thomas Caswell and Kyle Sunden, can be found at https://github.com/m atplotlib/data-prototype. Further research could investigate building new systems using this model, specifically libraries for visualizing domain specific structured data and domain specific artists. More research could also explore applying this model to visualizing high dimensional data, particularly building artists that take as input distributed data and artists that are concurrent. Developing complex systems could also be an avenue to codify how interactive techniques are expressed in this framework.

## VII. Conclusion

The toy example presented in section VI demonstrates that it is relatively straightforward to build working visualization library components using the construction described in section V. Since these components are defined with single record inputs, they can be implemented such that they are concurrent. The cost of building a new function using these components is sometimes as small as renaming fields, meaning the new feature is relatively easy to maintain. These new components are also a lower maintenance burden because, by definition, they are designed in conjunction with tests that verify that they are equivariant. These new components are also compatible with the existing library architecture, allowing for a slow iterative transition to components built using this framework. The framework introduced in this paper is a marriage of the ways the graphic and data visualization communities approach visualization. The graphic community prioritizes ? how input is translated to output, which is encapsulated in the artist $A$. The data visualization community prioritizes the manner in which that input is encoded, which is encapsulated in the separation of stages $Q \circ \nu$. Formalizing that both views are equivalent $A = Q \circ \nu$ gives library developers the flexibility to build visualization components in the manner that makes more sense for the domain without having to sacrifice the equivariance of the translation.

## Appendix A
## Summary

The topological spaces and functions introduced throughout this paper are summarized here for reference.

| | point/openset/base space | fiber space | total space |
|---|---|---|---|
| | location/subset/indices | record/fields | dataset type |
| Data | $k \in U \subseteq K$ | $r \in F$ | $E$ |
| Visual | $k \in U \subseteq K$ | $V \in P$ | $V$ |
| Graphic | $s \in W \subseteq S$ | $d \in D$ | $H$ |

TABLE I
Topological spaces introduced in subsection III-A

| | section | sheaf |
|---|---|---|
| | record at location | set of possible records for subset |
| Data | $\Gamma(K, E) \ni \tau : K \to F$ | $\mathcal{O}_{K,E} : U \to \Gamma(U, E \restriction_U)$ |
| Visual | $\Gamma(K, V) \ni \mu : K \to P$ | $\mathcal{O}_{K,V} : U \to \Gamma(U, V \restriction_U)$ |
| Graphic | $\Gamma(S, H) \ni \rho : S \to D$ | $\mathcal{O}_{S,H} : W \to \Gamma(U, H \restriction_W)$ |

TABLE II
Functions that associate topological subspaces with records, discussed in subsubsection III-A3 and subsection III-B

| axiom | applied to datasets and indexes |
|---|---|
| presheaf | given $\mathtt{index}_1 \subset \mathtt{index}_2$:<br>$\exists\ \mathtt{dataset}[\mathtt{index}_2]$<br>$\Rightarrow \mathtt{dataset}[\mathtt{index}_2][\mathtt{index}_1] = \mathtt{dataset}[\mathtt{index}_1]$<br>$\exists\ \mathtt{dataset}[\mathtt{index}_1] \not\Rightarrow \exists\ \mathtt{dataset}[\mathtt{index}_2]$ |
| locality | $\mathtt{dataset}^1[i] = \mathtt{dataset}^2[i]\ \forall\ i \in \mathtt{index}$<br>$\Rightarrow \mathtt{dataset}^1 = \mathtt{dataset}^2$ |
| gluing | $i = j$ and $\mathtt{dataset}^1[i] = \mathtt{dataset}^2[j]$<br>$\mathtt{dataset}^3 := \mathtt{dataset}^1[: i] \oplus \mathtt{dataset}^2[j :]$ |

TABLE III
Presheaf and sheaf constraints implemented by structure preserving data containers, discussed in subsection III-B

| | function | constraint |
|---|---|---|
| $s$ to $k$ | $\xi : W \to U$ | for $s \in W$ exists $k \in U$<br>s.t. $\xi(s) = k$ |
| graphic for $k$ | $\xi_* \rho : U \to \xi_* H \restriction_U$ | $\xi_* \rho(k)(s) = \rho(s)$ |
| record for $s$ | $\xi^* \tau : W \to \xi^* E \restriction_W$ | $\xi^* \tau(s) = \tau(\xi(s)) = \tau(k)$ |

TABLE IV
Functors between graphic and data indexing spaces subsection III-C

| changes | function | constraints, for all $k \in U$ |
|---|---|---|
| index | $\hat{\phi} : U \to U'$ | $\tau(k) = \tau(\hat{\phi}(k')) = \hat{\phi}^* \tau(k')$ |
| record | $\tilde{\phi} : \Gamma(U', \hat{\phi}^* E \restriction_U) \to \Gamma(U', \hat{\phi}^* E \restriction_U)$<br>$\tilde{\phi} : F \to F$ | $\lim_{x \to k} \tilde{\phi}(\tau(x)) = \tilde{\phi}(\tau(k))$<br>$\tilde{\phi}(\tau(k)) \in F$<br>$\tilde{\phi}(\mathtt{id}_F(\tau(k))) = \mathtt{id}_F(\tilde{\phi}(\tau(k)))$<br>$\tilde{\phi}(\tilde{\phi}(\tau(k))) = (\tilde{\phi} \circ \tilde{\phi})(\tau(k))$ |

TABLE V
Functions $\phi = (\hat{\phi}, \tilde{\phi})$ for modifying data records. Equivalent constructions can be applied to elements in visual and graphic sheaves, and these functions are distinguished through subscripts $\phi_E$, $\phi_V$ and $\phi_H$

| scale | operators | sample constraint |
|---|---|---|
| nominal | $=, \neq$ | $\tau(k_1) \neq \tau(k_2) \implies \tilde{\phi}(\tau(k_1)) \neq \tilde{\phi}(\tau(k_2))$ |
| ordinal | $<, \leqslant, \geqslant, >$ | $\tau(k_1) \leqslant \tau(k_2) \implies \tilde{\phi}(\tau(k_1)) \leqslant \tilde{\phi}(\tau(k_2))$ |
| interval | $+, -$ | $\tilde{\phi}(\tau(k) + C) = \tilde{\phi}(\tau(k)) + C$ |
| ratio | $*, /$ | $\tilde{\phi}(\tau(k) * C) = \tilde{\phi}(\tau(k)) * C$ |

TABLE VI
The record transformer $\tilde{\phi}$ must satisfy the constraints listed in Table V and $\tilde{\phi}$ must also respect the mathematical structure of $F$. This table lists examples of $\tilde{\phi}$ preserving one of the binary operators that are part of the definition of each of the Steven's measurement scale types[9]

. A full implementation would ensure that all operators that are defined as part of of $F$ are preserved.

| | function | constraints |
|---|---|---|
| artist<br>Data to Graphic | $A : \Gamma(K, E) \to \mathrm{Im}_A(S, H)$<br>$\mathrm{Im}_A(S, H) \subset \Gamma(S, H)$ | $\xi(S) = K$ |
| Encode | | |
| Decompose | | |

TABLE VII
ARTIST, VERIFICATION FUNCTIONS, AND CONSTRUCTION $A = Q \circ \nu$ INTRODUCED IN SECTION IV, AND SECTION V

color

| | function | constraint |
|---|---|---|
| artist | $A : \Gamma(K, E) \to \Gamma(S, H)$ | |
| lookup | $\xi : S \to K$ | |
| encoders | $\nu : \Gamma(K, E) \to \Gamma(K, V)$ | |
| compositor | $Q : \Gamma(K, V) \to \Gamma(S, V)$ | |

TABLE VIII
ARTIST, VERIFICATION FUNCTIONS, AND CONSTRUCTION $A = Q \circ \nu$ INTRODUCED IN SECTION IV, AND SECTION V

## APPENDIX B
## TRIVIAL AND NON-TRIVIAL BUNDLES

Generally, the distinguishing factor between a trivial bundle and a non-trivial bundle are how they are decomposed into local trivializations:

***trivial bundle*** is directly isomorphic to $K \times F$. For any choice of cover of $K$ by overlapping opensets, we can choose local trivializations such that all transition maps are identity maps.

***non-trivial bundle*** can not be constructed as $K \times F$. For any choice of local trivializations, there is at least one transition map that is not an identity [67].

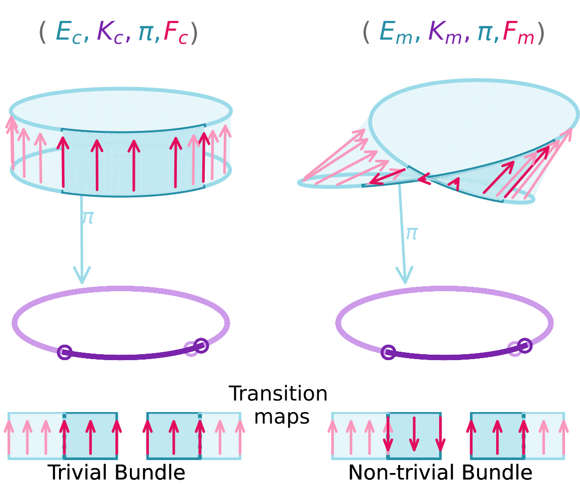



Fig. 16. The cylinder is a trivial fiber bundle; therefore it can be decomposed into local trivializations that only need identity maps to glue the trivializations together. The mobius band is a non-trivial bundle; therefore it can only be decomposed into trivializations where at least one transition map is not an identity map.

In the example in Figure 16, we use arrows $\uparrow$ to denote fiber alignments. In the cylinder case the fibers all point in the same direction, which illustrates that they are equal $\uparrow = \uparrow$. In the Möbius band case, while the fibers in an arbitrary local trivialization are equal $\uparrow = \uparrow$, the fibers at the twist are unequal but isomorphic $\uparrow \cong \downarrow$. The cylinder and mobius band can be decomposed to the same local trivializations, for example the fiber bundles in Figure 4 In the cylinder case, the fibers in the overlapping regions of the trivializations are equal $F_0 \restriction_{U_1 \cap U_2} = F_1 \restriction_{U_1 \cap U_2}$; therefore the transition maps at both intersections map the values in the fiber to themselves $r \to r$ . In the Möbius band case, while $F_0 \restriction_{(2\pi/5-\varepsilon, 2\pi/5+\varepsilon)} \to F_1 \restriction_{(2\pi/5-\varepsilon, 2\pi/5+\varepsilon)}$ can be chosen to be an identity map, the other transition map component $F_0 \restriction_{(-\varepsilon,\varepsilon)} \to F_1 \restriction_{(-\varepsilon,\varepsilon)}$ has to flip any section values. For example given $F_0 = \uparrow$ and $F_1 = \downarrow$, the transition map $r \mapsto -r$ maps each point from one fiber to the other $\uparrow \mapsto \downarrow$ such that any sections remain continuous even though the fibers point in opposite directions.

## APPENDIX C
## INTERNAL LIBRARY SPECIFICATION

As mentioned in subsection V-A, the internal types of visualization libraries can be defined using this model, which creates a consistent standard for developers writing new functions to target. These are the formal specifications of various aesthetic parameters in Matplotlib.

| $\nu_i$ | $\mu_i$ | $\mathtt{codomain}(\nu_i) \subset P_i$ |
|---|---|---|
| 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 IX
SOME OF THE P COMPONENTS OF THE V BUNDLES IN MATPLOTLIB COMPONENTS

## Appendix D
## Matplotlib Compatibility

As mentioned in section VI, one advantage of using this type of functional categorical approach to software design is that we can develop new components that can be incorporated into the existing code base. For matplotlib, we can use these functional artists by wrapping them in a very thin compatibility layer shim so that they behave like existing artists.

```python
class GenericArtist(martist.Artist):
    def __init__(self, artist:TopologicalArtist):
        super().__init__()
        self.artist = artist

    def compose_with_tau(self, section):
        self.section = section

    def draw(self, renderer, bounds, rate):
        for tau_local in self.section.query(bounds, rate):
            mu = self.artist.nu(tau_local)
            rho = self.artist.qhat(**mu)
            output = rho(renderer)
```

## Acknowledgment

Acknowledge all the actual people The authors would like to thank the anonymous reviewers who gave constructive feedback on an earlier version of this paper. The authors are also grateful to the various Matplotlib and Napari contributors, particularly Juan Nunez-Iglesias, and Nicolas Kruchten for their valuable feedback from the library developer perspective.

Hannah is also very grateful to Nicolas for the suggestion of augmented notation and to the nlab and wikipedia contributors who wrote clear explanations of many of the topics discussed in this paper.

This project has been made possible in part by grant number 2019-207333 and Cycle 3 grant number Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation

## References

[1] Z. Hu, J. Hughes, and M. Wang, "How functional programming mattered," *National Science Review*, vol. 2, no. 3, pp. 349–370, Sep. 2015, ISSN: 2095-5138. DOI: 10.1093/nsr/nwv042.

[2] J. Hughes, "Why Functional Programming Matters," *The Computer Journal*, vol. 32, no. 2, pp. 98–107, Jan. 1989, ISSN: 0010-4620. DOI: 10.1093/comjnl/32.2.98.

[3] A. Head, A. Xie, and M. A. Hearst, "Math augmentation: How authors enhance the readability of formulas using novel visual design practices," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI '22, New York, NY, USA: Association for Computing Machinery, 2022, ISBN: 978-1-4503-9157-3. DOI: 10.1145/3491102.3501932. [Online]. Available: https://doi.org/10.1145/3491102.3501932.

[4] L. Wilkinson, *The Grammar of Graphics* (Statistics and Computing), 2nd ed. New York: Springer-Verlag New York, Inc., 2005, ISBN: 978-0-387-24544-7.

[5] J. Bertin, *Semiology of Graphics : Diagrams, Networks, Maps*. Redlands, Calif.: ESRI Press, 2011, ISBN: 978-1-58948-261-6 1-58948-261-1.

[6] J. Mackinlay, "Automatic Design of Graphical Presentations," Stanford, 1987.

[7] G. Kindlmann and C. Scheidegger, "An Algebraic Process for Visualization Design," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2181–2190, Dec. 2014, ISSN: 1941-0506. DOI: 10.1109/TVCG.2014.2346325.

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

[9] S. S. Stevens, "On the Theory of Scales of Measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946, ISSN: 00368075, 10959203. JSTOR: 1671815. [Online]. Available: https://www.jstor.org/stable/1671815 (visited on 09/25/2020).

[10] W. A. Lea, "A formalization of measurement scale forms," *The Journal of Mathematical Sociology*, vol. 1, no. 1, pp. 81–105, 1971. DOI: 10.1080/0022250X.1971.9989789. [Online]. Available: https://doi.org/10.1080/0022250X.1971.9989789.

[11] M. A. Thomas, "Mathematization, Not Measurement: A Critique of Stevens' Scales of Measurement," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 2412765, Oct. 2014. DOI: 10.2139/ssrn.2412765.

[12] R. P. Grimaldi, *Discrete and Combinatorial Mathematics, 5/e*. Pearson Education, 2006.

[13] A. M. Pitts, *Nominal Sets: Names and Symmetry in Computer Science*. USA: Cambridge University Press, 2013, ISBN: 1-107-01778-5.

[14] E. H. Chi, "A taxonomy of visualization techniques using the data state reference model," in *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, Oct. 2000, pp. 69–75. DOI: 10.1109/INFVIS.2000.885092.

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

[16] E. Riehl, *Category Theory in Context*. Dover Publications, Nov. 16, 2016.

[17] D. M. Butler and S. Bryson, "Vector-Bundle Classes form Powerful Tool for Scientific Visualization," *Computers in Physics*, vol. 6, no. 6, p. 576, 1992, ISSN: 08941866. DOI: 10.1063/1.4823118.

[18] D. M. Butler and M. H. Pendley, "A visualization model based on the mathematics of fiber bundles," *Computers in Physics*, vol. 3, no. 5, p. 45, 1989, ISSN: 08941866. DOI: 10.1063/1.168345.

[19] T. Munzner, "What: Data Abstraction," in *Visualization Analysis and Design*, ser. AK Peters Visualization

Series, A K Peters/CRC Press, Oct. 2014, pp. 20–40, ISBN: 978-1-4665-0891-0. DOI: 10.1201/b17511-3.

[20] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, vol. 5, no. 2, pp. 110–141, Apr. 1986, ISSN: 0730-0301. DOI: 10.1145/22949.22950.

[21] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A system for query, analysis, and visualization of multidimensional relational databases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52–65, Jan. 2002, ISSN: 1941-0506. DOI: 10.1109/2945.981851.

[22] P. Hanrahan, "VizQL: A language for query, analysis and visualization," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06, New York, NY, USA: Association for Computing Machinery, 2006, p. 721, ISBN: 1-59593-434-0. DOI: 10.1145/1142473.1142560.

[23] J. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1137–1144, Nov. 2007, ISSN: 1941-0506. DOI: 10.1109/TVCG.2007.70594.

[24] K. Wongsuphasawat, "Navigating the Wide World of Data Visualization Libraries (on the web)," presented at the Outlier Conf, 2021. [Online]. Available: https://www.slideshare.net/kristw/navigating-the-wide-world-of-data-visualization-libraries.

[25] K. Wongsuphasawat, "Navigating the Wide World of Data Visualization Libraries," *Nightingale*, Sep. 22, 2020. [Online]. Available: https://nightingaledvs.com/navigating-the-wide-world-of-data-visualization-libraries/.

[26] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, May 2007, ISSN: 1558-366X. DOI: 10.1109/MCSE.2007.55.

[27] M. Bostock, V. Ogievetsky, and J. Heer, "D$^3$ Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011, ISSN: 1941-0506. DOI: 10.1109/TVCG.2011.185.

[28] M. D. Hanwell, K. M. Martin, A. Chaudhary, and L. S. Avila, "The Visualization Toolkit (VTK): Rewriting the rendering code for modern graphics cards," *SoftwareX*, vol. 1–2, pp. 9–12, Sep. 2015, ISSN: 23527110. DOI: 10.1016/j.softx.2015.04.001.

[29] B. Geveci, W. Schroeder, A. Brown, and G. Wilson, "VTK," *The Architecture of Open Source Applications*, vol. 1, pp. 387–402, 2012.

[30] J. Heer and M. Agrawala, "Software design patterns for information visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 853–860, 2006. DOI: 10.1109/TVCG.2006.178.

[31] H. Wickham, *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016, ISBN: 978-3-319-24277-4.

[32] A. Satyanarayan, K. Wongsuphasawat, and J. Heer, "Declarative interaction design for data visualization," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, Honolulu Hawaii USA: ACM, Oct. 2014, pp. 669–678, ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647360.

[33] J. VanderPlas, B. Granger, J. Heer, *et al.*, "Altair: Interactive Statistical Visualizations for Python," *Journal of Open Source Software*, vol. 3, no. 32, p. 1057, Dec. 2018, ISSN: 2475-9066. DOI: 10.21105/joss.01057.

[34] N. Sofroniew, T. Lambert, K. Evans, *et al.*, *Napari/napari: 0.4.5rc1*, version v0.4.5rc1, Zenodo, Feb. 2021. DOI: 10.5281/zenodo.4533308. [Online]. Available: https://doi.org/10.5281/zenodo.4533308.

[35] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, "NIH Image to ImageJ: 25 years of image analysis," *Nature Methods*, vol. 9, no. 7, pp. 671–675, Jul. 2012, ISSN: 1548-7105. DOI: 10.1038/nmeth.2089.

[36] S. Studies, *Culturevis/imageplot*, Jan. 15, 2021. [Online]. Available: https://github.com/culturevis/imageplot (visited on 02/11/2021).

[37] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," presented at the International AAAI Conference on Weblogs and Social Media, ser. Proceedings of the International AAAI Conference on Web and Social Media, vol. 3, 2009.

[38] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[39] V. Wiels and S. Easterbrook, "Management of evolving specifications using category theory," in *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No.98EX239)*, 1998, pp. 12–21. DOI: 10.1109/ASE.1998.732561.

[40] B. A. Yorgey, "Monoids: Theme and variations (functional pearl)," *SIGPLAN Not.*, vol. 47, no. 12, pp. 105–116, Sep. 2012, ISSN: 0362-1340. DOI: 10.1145/2430532.2364520. [Online]. Available: https://doi.org/10.1145/2430532.2364520.

[41] P. Vickers, J. Faith, and N. Rossiter, "Understanding Visualization: A Formal Approach Using Category Theory and Semiotics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 6, pp. 1048–1061, Jun. 2013, ISSN: 1941-0506. DOI: 10.1109/TVCG.2012.294.

[42] D. I. Spivak, "Databases are categories," slides, presented at the Galois Connections (Galois Inc.), Jun. 3, 2010.

[43] D. I. Spivak. "Simplicial databases." (2009), [Online]. Available: https://arxiv.org/abs/0904.2012, prepublished.

[44] E. Spanier, *Algebraic Topology* (McGraw-Hill Series in Higher Mathematics). Springer, 1989, ISBN: 978-0-387-94426-5.

[45] E. W. Weisstein. “Open Set,” MathWorld–A Wolfram Web Resource. (), [Online]. Available: https://mathworld.wolfram.com/ (visited on 08/19/2022).

[46] T.-D. Bradley, *Topology vs. "A Topology" (cont.)* [Online]. Available: https://www.math3ma.com/blog/topology-vs-a-topology.

[47] R. Brown, *Topology and Groupoids*. 2006, ISBN: 1-4196-2722-8. [Online]. Available: http://groupoids.org.uk/pdffiles/topgrpds-e.pdf.

[48] J. R. Munkres, *Elements of Algebraic Topology*. Menlo Park, Calif: Addison-Wesley, 1984, ISBN: 978-0-201-04586-4.

[49] F. W. Lawvere and S. H. Schanuel, *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, 2009.

[50] S. Mac Lane, *Categories for the Working Mathematician*. Springer Science & Business Media, 2013, vol. 5, ISBN: 1-4757-4721-7.

[51] B. Fong and D. I. Spivak, *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*, 1st ed. Cambridge University Press, Jul. 2019, ISBN: 978-1-108-66880-4 978-1-108-48229-5 978-1-108-71182-1. DOI: 10.1017/9781108668804.

[52] J. D. Ullman and Jennifer. Widom, *A First Course in Database Systems*. Upper Saddle River, NJ: Pearson Prentice Hall, 2008, ISBN: 0-13-600637-X 978-0-13-600637-4.

[53] R. Brüggemann and G. P. Patil, *Ranking and Prioritization for Multi-indicator Systems: Introduction to Partial Order Applications*. Springer Science & Business Media, Jul. 2011, ISBN: 978-1-4419-8477-7.

[54] R. W. Ghrist, *Elementary Applied Topology*. Createspace Seattle, 2014, vol. 1.

[55] *Cairographics.org*. [Online]. Available: https://www.cairographics.org/ (visited on 02/17/2021).

[56] T.-D. Bradley. “What is a Functor? Definitions and Examples, Part 2,” Math3ma. (), [Online]. Available: https://www.math3ma.com/blog/what-is-a-functor-part-2 (visited on 10/07/2021).

[57] T.-D. Bradley, J. Terilla, and T. Bryson, *Topology : A Categorical Approach*. MIT Press, 2020, ISBN: 978-0-262-35962-7 0-262-35962-6. [Online]. Available: https://topology.mitpress.mit.edu/.

[58] nLab authors, *Presheaf*, Oct. 2021.

[59] M. J. Baker. “EuclideanSpace: Maths - Sheaf.” (), [Online]. Available: https://www.euclideanspace.com/maths/topology/sheaf/ (visited on 01/06/2022).

[60] G. Harder and K. Diederich, *Lectures on Algebraic Geometry I: Sheaves, Cohomology of Sheaves, and Applications to Riemann Surfaces* (Aspects of Mathematics). Vieweg+Teubner Verlag, 2008, ISBN: 978-3-8348-9501-1.

[61] R. A. Becker and W. S. Cleveland, “Brushing Scatterplots,” *Technometrics : a journal of statistics for the physical, chemical, and engineering sciences*, vol. 29, no. 2, pp. 127–142, May 1987, ISSN: 0040-1706. DOI: 10.1080/00401706.1987.10488204.

[62] A. Quint, “Scalable vector graphics,” *IEEE MultiMedia*, vol. 10, no. 3, pp. 99–102, Jul. 2003, ISSN: 1941-0166. DOI: 10.1109/MMUL.2003.1218261.

[63] T.-D. Bradley. “What is a Natural Transformation? Definition and Examples,” Math3ma. (), [Online]. Available: https://www.math3ma.com/blog/what-is-a-natural-transformation (visited on 10/12/2021).

[64] B. Milewski, *Category Theory for Programmers*. Bartosz Milewski, Jan. 1, 2019.

[65] M. Krstajić and D. A. Keim, “Visualization of streaming data: Observing change and context in information visualization techniques,” in *2013 IEEE International Conference on Big Data*, 2013, pp. 41–47. DOI: 10.1109/BigData.2013.6691713.

[66] D. Nekrasovski, A. Bodnar, J. McGrenere, F. Guimbretière, and T. Munzner, “An evaluation of pan & zoom and rubber sheet navigation with and without an overview,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’06, New York, NY, USA: Association for Computing Machinery, 2006, pp. 11–20, ISBN: 1-59593-372-7. DOI: 10.1145/1124772.1124775.

[67] A. Hatcher, *Algebraic Topology*. Cambridge University Press, 2002.

[68] J. Bertin, “II. The Properties of the graphic system,” in *Semiology of Graphics*, Redlands, Calif.: ESRI Press, 2011, ISBN: 978-1-58948-261-6 1-58948-261-1.

[69] Z. Qu and J. Hullman, “Keeping multiple views consistent: Constraints, validations, and exceptions in visualization authoring,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 468–477, Jan. 2018, ISSN: 1941-0506. DOI: 10.1109/TVCG.2017.2744198.

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

[71] J. Hunter and M. Droettboom. “The Architecture of Open Source Applications (Volume 2): Matplotlib.” (), [Online]. Available: https://www.aosabook.org/en/matplotlib.html (visited on 01/28/2021).

[72] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[73] J. Reback, W. McKinney, jbrockmendel, *et al.*, *Pandas-dev/pandas: Pandas 1.0.3*, version v1.0.3, Zenodo, Mar. 2020. DOI: 10.5281/zenodo.3715232. [Online]. Available: https://doi.org/10.5281/zenodo.3715232.

[74] S. Hoyer and J. Hamman, “Xarray: ND labeled arrays and datasets in Python,” *Journal of Open Research Software*, vol. 5, no. 1, 2017.


**Hannah Aizenman** Biography text here.

**Thomas Caswell** Biography text here.

1795 **Michael Grossberg** Biography text here.