Topological Equivariant Artist Model for Visualization Library Architecture

Hannah Aizenman, Thomas Caswell, and Michael Grossberg, *Member, IEEE,*

Abstract—The abstract goes here.

Index Terms—

1 Introduction

Visualizations are expected to be a faithful translation of data, meaning that inherent structure in the data should be present in the visualization; therefore the components of visualization software libraries that translate data to graphics are also expected to be structure preserving. This inherent structure can be quite complex, such as when the data is high dimensional, multivariate, deeply nested, or encoded in a distributed manner. We propose that we can rigorously specify what structure the library components are expected to preserve using category theory. We also propose that these specifications can generalize to most structure types by extending previous work on encoding inherent structure in a consistent manner using mathematical structures from algebraic topology. These algebraic structures can be translated into analogous programmatic types for encoding structure, while the use of category theory facilitates developing a functional design framework that describes function composition in terms of functional algebraic operations. The typing system and composition inherent in a functional design encourage library developers to build complex visualization components from simple verifiable parts that have few side effects [1], [2]. These categorically specified components could be built as a standalone library and integrated into existing libraries because they are inherently self-contained. Furthermore we see the application of these ideas in low level visualization library design as our motivation, and we hope these ideas will help shape the reorganization of critical data visualization libraries, such as Matplotlib. The contribution of this paper is a mathematical framework for describing structure preserving visualization components in a manner that is generalizable to different types of inherent structure. This framework also provides guidance for the construction and testing of structure preserving visualization library components.

- *H. Aizenman and M. Grossberg are with the department of Computer Science, City College of New York.*
  E-mail: haizenman@ccny.cuny.edu, mgrossberg@ccny.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.

2 Related Work

We build on the extensive work codifying the structure of visualization data to develop a more generalizable method for expressing the structure that visualizations are expected to preserve. Structure preservation is important because it means visualzations are easier to understand [3] and ensures that the visual representations are not distortions of the data [4]. Broadly, previous work describes structure as a combination of the topological properties of the data, the mathematical structure of the data fields, and equivalent or compatible changes to data and graphics. We propose that a unified abstraction model can guide developers in building structure aware composable software library components with more consistent interfaces.

2.1 Topological Structure

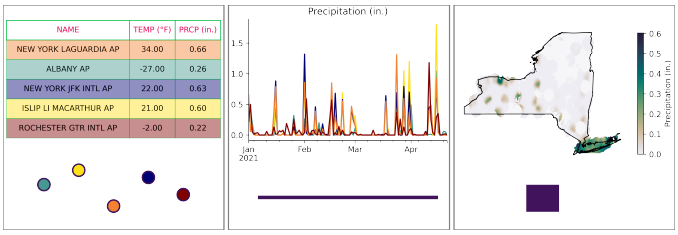


Fig. 1. A a topological representation of the data (in purple) is a simplified representation of a view of how the records in a dataset are connected. The table can be viewed as a set of independent records; therefore it has a 0D continuity represented by the disconnected dots. These weather records are also samples of continuous measurements at each location, which can be modeled as the time series measurements being samples on a 1d continuous interval and the geospatial station measurements as sparse samples on a 2D continuous space.

There is an assumption in visualization that the connectivity of the elements in a dataset are preserved in the visual representation; roughly topology provides a method for encoding this connectivity [5]. Bertin gives the examples of discrete data values should be represented by discrete entities such as points, quantitative continuous data by lines, and surfaces through area marks [6]. In Figure 1 the records of the GHCN [7] weather station table can be considered disconnected from each other and therefore

have a 0D continuity. This means each distinct row can be reduced to a discrete point and each row will be encoded as a discrete visual element, such as a standalone scatter point. Each station can also be considered a time series of weather observation since temperature and pressure are sampled from a 1D continuous measurement space. The connectivity of these observations can be modeled as an interval and the data is encoded as a line since that visual element is also 1D continuous. The measurements at set of stations on any given day is a sparse sampling of a 2D geospatial grid and the connectivity can be modeled as a 2D plane. Instead of interpolation, which is how continuity was preserved in the timeseries plot, in this instance the connectivity is preserved by plotting the stations in the map such that their relative distances are preserved. Encoding connectivity using the formalism of topological spaces allows us to encode the connectivity of the data in a uniform manner such that we can then generalize the preservation of that connectivity in the visual representation rather than constructing a condition specifically for each type of topological structure.

Visual algorithms assume the topology of their input data, as described in taxonomies of visualization algorithms Chi [8] and by Troy and Möller [9]. 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 1. 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. Since topological continuity is generalizable, we propose the construction of topology types such that data can explicitly state its topology type and visual algorithms can check that this type matches their assumptions. The typing system proposed in this paper builds on Butler's proposal of using a mathematical structure called fiber bundles as an abstract data representation in visualization [10], [11]. We extend Butler's work because he discusses how bundles can be used to encode data in a consistent manner for the topological and field structures that are common in visualization. We sketch out fiber bundles in subsection 3.1, but his work provides a thorough introduction to bundles for visualization practitioners.

2.2 Field Type Structure

As with topology, Bertin [6] codified the expectation that a data value and its visual encoding are expected to have compatible structure. The structure on values has traditionally been described using the measurement scales-i.e. nominal, ordinal, interval, ratio - which Steven's classified by their mathematical group structure [**?**] and others have formally expanded to include more types [12], [13]. Generally the structure on each field of the dataset is preserved separately, i.e. each column in a table is mapped to a distinct encoding. Two mathematical concepts for describing a function that preserves structure are *equivariance* and *homomorphism*. An *equivariant* function preserves symmetric group structure on its input and output, while a *homomorphic* function f preserves binary operations on input and output of the same algebraic type. Group structure and binary operations are both describing families of functions or operations that can be applied to data and visual encodings. A group G is a set with an operator $\circ$. The set contains an identity element $e \in G$ and the operator $\circ$ is closed, associative and invertible. Applying elements of a set, such as a group, to another set, such as data X, is called an *action*.

Definition 2.1. [14], [15] An **action** [1] of G on X is a function $act : G \times X \to X$. 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 $f, g \in G$.

In **??**, actions on data and compatible actions on graphics define structure in terms of testing whether a visualization is equivariant. Given a group G that acts on input X and output Y and a function $f : X \to Y$

Definition 2.2. f is **equivariant** when $f(g(x)) = g(f(x))$ for all g in G and for all x in X [17], [18].

Definition 2.3. f is **homomorphic** when $f(x_1 \oslash x_2) = f(x_1) \oslash f(x_2)$ for every pair x_1, x_2 in X [15].

For example, nominal measurements are permutable, meaning that they are acted on by elements of a permutation group. For the visual transformation to be equivariant, that nominal measurement must be mapped to a visual encoding that is also permutable, such as distinct marker shapes or color hues. We evaluate whether the encoding is homomorphic when the visual encoding does not have a group structure. For example, partial orders are not invertible; therefore a visual encoding of partially ordered data is structure preserving when the ordering is mapped to equivalently ordered visual encodings.

The notion that visual encodings should be *homomorphic* was proposed by Mackinlay [19] in his specification of *A Presentation Tool* and the idea that visual transforms are *equivariant* underlies Kindlemann and Scheidegger [20]'s Algebraic Visualization Design. Kindlemann and Scheidegger propose three specific types of structure preservation: that the visualization should not change if the data representation (i.e. the data container) changes, that data maps unambiguously to visual elements [21], and that changing the data should correspond to changes in the visualization in a perceptually significant manner. In subsection 3.2 we introduce a uniform abstract data representation layer, the expectation of unambiguous mappings is expressed in subsection 3.3, and the definition of equivariance introduced in subsubsection 4.1.2 includes the correspondence between data and visual changes. In this work, we formally specify the conditions necessary to build structure preserving components using category theory, because it provides a rich language for expressing the structure of objects, functions between objects, and the constraints that must hold for these functions to compose [22], [23]. A brief visualization oriented introduction to category theory is in Vickers et al [24], but they are applying category theory to semantic concerns about visualization design rather than library architecture.

[1] Throughout this paper, we augment the math with color to group conceptually similar objects and functions [16], for example actions and sets of actions. This color coding carries through to the figures.

2.3 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 [25]. While some visualization tools aim to automate the pairing of data with structure preserving visual representations, such as Tableau [26], [27], [28], 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 [29] 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 [30] and D3 [31] the connectivity assumptions lead to very different interfaces; for example in Matplotlib methods for updating data and parameters for controlling aesthetics differ between plots. While VTK [32], [33] provides a language for expressing the topological properties of the data, and therefore can embed that information in its visual algorithms, it does so in a non-uniform manner. On the other hand, domain specific libraries are designed with the assumption of continuities that are common in the domain [34], and therefore can somewhat restrict the interface to choices that that are appropriate for the domain. For example, a tabular topological structure of discret rows, as illustrated in Figure 1, is assumed by A Presentation Tool [25], [25] and grammar of graphics [5] and the ggplot [35], vega [36], and altair [37] libraries built on these frameworks. Image libraries such as Napari [38] and ImageJ [39] and its humanities ImagePlot [40] plugin assume that the input is 2D continuous. Networking libraries such as gephi [41] and networkx [42] assume a graph-like structure. By assuming the structure of their data, these domain specific libraries can provide more cohesive interfaces for a much more limited set of visualization algorithms than the building block libraries offer.

3 Formal Properties of Data & Graphics

In this section, we propose a mathematical abstraction of the data input and graphic prerendered output. This mathematical abstraction has a robust language for expressing 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.

3.1 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 bundles as abstract visualization data type [10], [11] by incorporating Spivak's methodology for encoding named data types from his fiber bundle representation of relational databases [43], [44]. We build on this work to describe how to encode the connectivity of the data as a topological space, separately encode the fields as their own topological space with a typing system, and express the mappings between these two spaces.

Definition 3.1. A **fiber bundle** (E, K, π, F) is a structure with topological spaces E, F, K and bundle projection map $\pi : E \to K$ [45], [46].

$$\begin{array}{ccc} F & \hookrightarrow & E \\ & & \downarrow \pi \\ & & K \end{array} \qquad (1)$$

A continuous surjective map π is a **bundle projection** map when

1) the fiber space F is the preimage of the projection function π at a point k in the base space K such that $F_k = \pi^{-1}(k)$. All fibers in a bundle are isomorphic such that $F \cong F_k$ for all points $k \in K$.
2) there is an open neighborhood U_k surrounding each point in the base space $k \in U_k \subset K$ such that the total space E over the neighborhood, denoted $E \restriction U$[2], is locally trivial. The condition for local triviality is that $E \restriction_U = U \times F = \pi^{-1} \restriction_U$

Definition 3.2. A **section** $\tau : K \to E$ over a fiber bundle is a smooth right inverse of π such that $\pi(\tau(k)) = k$ for all $k \in K$

Local triviality $E \restriction_{U_k} = U_k \times F$ means that there is always a subset of the base space over which the fibers are identical. A bundle can be subsetted into regions such that the bundled can be reconstructed by gluing the regions back together via transition maps on the border of each region. These borders are fibers, and in a trivial bundle transition maps are identity maps because fibers are identical $F = F_k$ for all points $k \in \mathtt{dbase}$. In a non-trivial bundle fibers are isomorphic $F \simeq F_k$ for all points $k \in K$, which means that there is at least one transition map that is not an identity [48].

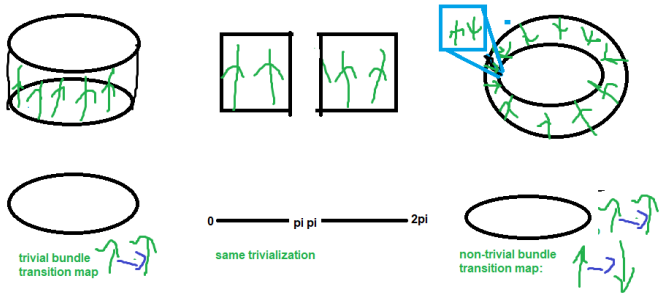


Fig. 2. A fiber bundle with a cylinderical total space, which is a trivial bundle, and bundle with a mobius band total space, which is non-trivial, can be covered by the same set of local trivializations. For example, they can both be decomposed into two planes over two intervals $(-\epsilon, \pi + \epsilon), (\pi - \epsilon, 2\pi + \epsilon)$. The transition maps that transforms this set of local trivializations into a cylinder are two identity map aligning two *up* fibers, while the transition maps that transform this set into a mobius bundle are an identity map and a map that aligns an *up* fiber with a *down* fiber. in final may change this to something that's not half half to make the arbitraryness more clear/break it up into smaller pieces

As shown in Figure 2, a trivial bundle with a cylinderical total space and a non-trivial bundle with mobius band total

[2]The symbol $\restriction$ is the restriction operator [47] defined in `amssymb`. For example $\pi^{-1} \restriction U$ is the inverse function π^{-1} defined only on the points in U. Since $\pi^{-1}(K) = E$, we use the shorthand $E \restriction_U := \pi^{-1} \restriction_U$

spacee can be covered with the same local trivializations; the distinguishing factor between these bundles is that the transition maps on these trivializations differ. The fibers in the cylinder bundle align in the same direction, *up*, such that the transition maps that assemble the local trivializations into the cylinderical total space are all identity maps aligning two *up* fibers. In the mobius bundle, there is twist where the fibers align in different directions, shown in the inset in Figure 2. This twist is not present in either local trivialization because, by definition, fibers need to be aligned in local trivializations; therefore the transition map that aligns the fiber at the overlap between $(-\epsilon, \epsilon)$ must account for this flip by aligning *up* fibers *down* fibers. In this example, the transition map on the $(\pi - \epsilon, \pi + \epsilon)$ is an identity map since the fibers are aligned in the same direction on that section of the mobius band. add something in the figure highlighting this part of the cylinder and the mobius band, possibly add in a section

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, and the fiber space can encode the data types of the record fields of the dataset. The map π expresses the formal binding between having a typed data field, discussed in subsubsection 3.1.2, and a corresponding point in the topological structure, which is discussed in subsubsection 3.1.1. Fiber bundles are a good abstract data representation for visualization because the field type and topological structure are unconstrained in terms of dimensionality and the only conditions that must be satisfied are that every point in the base space has a corresponding field of values and the field types must be the same for every point in the base space. We propose that modeling data as sections provides a way to encapsulate topological and field structure in a uniform dimension and type independent manner, as discussed in subsubsection 3.1.3.

3.1.1 *Topological Structure: Base Space* K

We encode the topological structure of the data as the base space K of the fiber bundle since the base space acts as indexing space where every point in the base space has a corresponding fiber space $\pi^{-1}(k) = F_k$. The base space K is a topological space $(K, \mathcal{T}_K)$, which means it consists of the set of points $k \in K$ and topology $\mathcal{T}_K$ [49]. A topology $K := (K, \mathcal{T}_K)$ is an axiomatic way of defining a topological structure [50] as a collection of subsets, called open sets [3], of that structure.

Definition 3.3. A **topology** $\mathcal{T}$ on a space X is a collection of open sets U of X. This collection has the properties that the empty set $\varnothing$ and K are in $\mathcal{T}$, the union of open sets in $\mathcal{T}$ is also in $\mathcal{T}$, and any finite intersection of open sets in $\mathcal{T}$ is also in $\mathcal{T}$. [50]

We propose that the topology on K can act as a mathematical abstraction for the indexing space of a dataset because the properties of a topology are the same as would be expected of an index space, namely that the space has a mathematical structure, that the space can be subsetted, and that continuity is preserved when sets of indices are merged. For example, in Figure 1, a topology on the line would break the line up into subsets, and those subsets could only be joined in ways where they cover the line in the same order in which it was broken up. The base space of a fiber bundle is a quotient topology [49], [53], meaning that it divides the topological space into the smallest possible (largest number of) open sets such that π remains a continuous function. This means that the topology can be defined to have a resolution equal to the number of indicies in a dataset or at whatever resolution of continuois function is required for a given task.

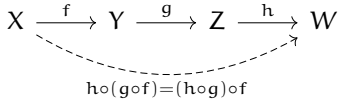
The topological structure of the data can be expressed programmatically by constructing data types that encapsulate the class of topological structure-.i.e point, line, plane, network, etc. We propose that these data types can be formally specified as objects of a category.

Definition 3.4. An **category** $\mathcal{C}$ consists of the following *data*:

1) a collection of *objects* $X\mathbf{ob}(\mathcal{C})$
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* $X \xrightarrow{\mathrm{id}_X} X$ 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 $X \xrightarrow{f} Y$, $f \circ \mathrm{id}_x = f = \mathrm{id}_y \circ f$
2) *associativity:* if any three morphisms f, g, h are composable,



then they are associative such that $h \circ (g \circ f) = (h \circ g) \circ f$ [54], [55], [56], [57].

We encapsulate the topological structure of the data as a category $\mathcal{K}$. 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 $U_i \subseteq U_j$ [55]. 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 datatypes as $\mathcal{K}$, we can express that these are the properties that are required as part of the implementation of the data type objects.

3.1.2 *Data Field Types: Fiber Space* F

As mentioned in subsection 2.2, 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 [43], [44]. 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

[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. [51], [52]

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 **DT** is a set of data types $T \in \mathbf{DT}$ and the total space $\mathscr{U}$ is the disjoint union of the domains of each type

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

such that each element x in the domain $\pi^{-1}(T)$ is one possible value of an object of type T [44]. For example, if $T = \texttt{int}$, 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 [58]. According to Spivak, a *schema* consists of a pair (C, σ) where C is the set of field names and $\sigma : C \to \mathbf{DT}$ is a function from field name to field data type [44]. The function σ is composed with π 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 3.5. A **record** is a function $r : C \to \mathscr{U}_\sigma$ and the set of records on π_σ 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 set of all data records

$$F \coloneqq \{r : C \to \mathscr{U}_\sigma \mid \pi_\sigma(r(C)) = C \text{ for all } C \in C\} \tag{2}$$

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 accesed by field name, since those are sections on $\mathcal{U}_\sigma$. Furthermore, since domains $\mathscr{U}_\sigma$ of types are a topological 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 $\tilde{\phi} \in \mathrm{Hom}(F, F)$. The fiber category $\mathcal{F}$ is a monoidal category, meaning that it is a category equipped with a bifunctor $\otimes : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$. The bifunctor provides a method for combining fibers, thereby allowing us to express fields that contain multityped values. For example, wind can be represented as two fields $F_{\texttt{speed}} \times F_{\texttt{direction}}$ or a composite fiber field $F_{\texttt{speed}} \otimes F_{\texttt{direction}} = F_{\texttt{wind}}$. The $\otimes$ encapsulates both the sets associated with each fiber $\mathbb{R} \times \mathbb{R} = \mathbb{R}^2$ and the morphisms associated with each functor $(\tilde{\phi}_{\texttt{speed}}, \tilde{\phi}_{\texttt{direction}}) = \tilde{\phi}_{\texttt{wind}}$. Defining the field schema as a monoidal category allows us to formally express the structure of the field, as defined by its sets and functions, at different levels of composition as needed by different visualization tasks.

3.1.3 Data: Section

We encode data as a section τ of a bundle because this allows us to incorporate the topology and field types in the data definition. We can define these section functions locally, meaning that the section is (piece-wise) continuous over a specific open subset U of K

$$\Gamma(U, E \restriction_U) \coloneqq \{\tau : U \to E \restriction_U \mid \pi(\tau(k)) = k \text{ for all } k \in U\} \tag{3}$$

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.

In a trivial bundle, the total space is the product of the fiber and base space $E = K \times F$. This allows us to define global sections $\tau : K \to F \in \Gamma(K, F)$ which we translate into a data signature of the form

$$\texttt{dataset} : \texttt{topology} \to \texttt{field} \tag{4}$$

where $\tau = \texttt{dataset}$, $K = \texttt{topology}$ and $F = \texttt{fields}$. This type signature provides a method of explicitly stating the topology and field type of the data and generalizes to almost any topology and fiber type, provided that the total space is trivial.

When the total space 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

$$\begin{aligned} &\{\texttt{data-subset} : \texttt{topology} \to \texttt{fields} \\ &\qquad \text{s. t. } \texttt{data-subset} \subset \texttt{dataset}\} \end{aligned} \tag{5}$$

The subsets of the fiber bundle and the transition maps between these subsets are encoded in an atlas [59] and the notion of an atlas can be incorporated into the data container, as discussed in subsection 3.2.

3.1.4 Example

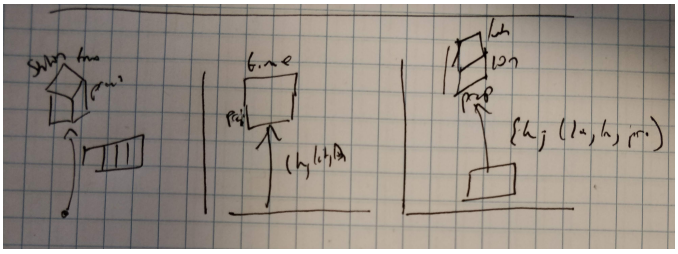


Fig. 3. The table from Figure 1 has a 3 dimensional fiber (name, temperature, precipitation), a 0D base space, and each row is a section. Each time series is a section of a bundle with a 2D fiber (time, precipitation) and a 1D base spac encoding temporal continuity. Each location of the rain map is a section of a bundle with a 2D plane base encoding spatial continuity and a 3D fiber space encoding (latitude, longitude, precipitation)

In Figure 3, the base space K acts as an indexing space into the fiber space F. In the bundle encoding of the table, the indexing space is arbitrarily numbered keys; in the time series bundle, the base space is the interval [0,1]; and the map is sparse samples from a continuous space $[0,1]^2$ In contrast to a notion of a semantic binding between indexing space and field values, as proposed by Munzner [60], the fields describing the continuity are part of the fiber. For example, the time is a fiber in the timeseries and the latitude and longitude are part of the map's fiber. This separation between connectivity and what it is called means the time or location can change units without a change to structure. It also provides a way to express data that may seem continuous but isn't, for example independent measurements over time. The data in Figure 3 comes from the same dataset; therefore we know that the bundles share connectivity and fiber space. It is expected that shared components be translated to visual elements in a consistent manner [61], and in subsection 4.2 we introduce operators for expressing which components are shared and how to verify that they have been mapped into visual elements in a consistent manner.

3.1.5 *Uniform Abstract Graphic Representation*

One of the advantages of fiber bundles is that they are general enough that we can also encode the output of a visual algorithm as a bundle. This allows us to use the same structure to express the properties of data and the graphic that must be symmetric to the data in an equivariant (subsection 2.2) transformation. 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.

$$D \hookrightarrow H \xrightarrow{\pi} S \tag{6}$$

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 cairo [62]. 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 generating functions as sections ρ of the graphic bundle

$$\Gamma(W, H\restriction_W) \coloneqq \left\{\rho : W \to H\restriction_W \;\middle|\; \pi(\rho(s)) = s \text{ for all } s \in W\right\} \tag{7}$$

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 to a renderer to generate graphics.

In Figure 4, the section function ρ maps into the fiber for a simplified 2D RGB infinite resolution prerender space and returns the $\{x, y, r, g, b\}$ values of a pixel in an infinite resolution space. In Figure 4 these pixels are approximated as the small orange and black 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. While Figure 4 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 layers based on D_z and D_a.

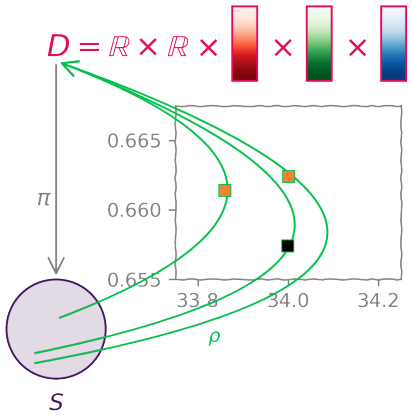


Fig. 4. For a 2D display, a section ρ maps from each point s into a fiber D that encodes an RGB prerender space with infinite resolution $\mathcal{R}^2$. Each tiny colored box is an approximation of the return value of the same section function ρ evaluated on different points $s \in S$ in the base space. add alpha and z channels and very faded out marker point

3.2 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 [59]. This abstraction facilitates representational invariance, as introduced by Kindlemannn and Scheidegger [20], since the container level is uniformally specified as satisfying sheaf constraints. These constraints generalize to data that is subsetted, distributed, streaming, and on-demand.

We can mathematically encode that we expect data containers to preserve the underlying continuity of the indexing space and the mappings beween 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 [55].

Definition 3.6. [50], [63] 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*: $F(\mathrm{id}_C(C)) = \mathrm{id}_D(F(C))$
- *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.

$$\mathcal{O}_{K,E} : U \to \Gamma(U, E \restriction_U) \tag{8}$$

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

Definition 3.7. 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 $\mathbf{Set}$ [46], [64].

Contravariance means that the morphisms between the input openset objects go in the opposite direction from the morphisms between the output set objects. The presheaf is contravariant because the inclusion morphisms between input objects

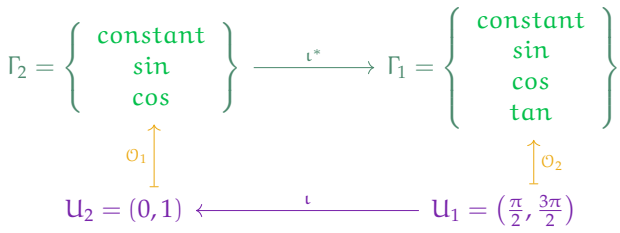
$$\iota : U_1 \to U_2$$

are defined such that they correspond to the partial ordering $U_1 \subseteq U_2$, but the restriction morphisms ι^* between the sets of sections

$$\iota^* : \Gamma(U_2, E \restriction_{U_2}) \to \Gamma(U_1, E \restriction_{U_\epsilon 1})$$

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 $\Gamma_i := \Gamma(U_i, E \restriction_{U_i})$.

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 Γ_1, Γ_2 that are continuous over that interval:



The constraints of a presheaf functor are that since the $constant, sin, cos$ functions are defined over the interval $[0, 1]$, these functions must also be continuous over the subinterval $\left(\frac{\pi}{2}, \frac{3\pi}{2}\right)$; therefore the sections in Γ_2 must also be included in the set of sections over the subspace Γ_1. The 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.

Definition 3.8. [46], [65] 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 over the same set of open sets $\tau^a|_U = \tau^b|_U$.
- *gluing* the union of sections defined on specific open sets is equivalent to one big section over the union of spaces $\tau|_{U_i \cup U_j} = \tau^i|_{U_i} \cup \tau^j|_{U_j}$ if these 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 locality axiom asserts that the glued section function is equivalent to a function over the union if they evaluate to the same values. 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. Generally, the sheaf asserts the expectation that the data container is implemented such that the connectivity between the opensets (indexing subspaces) is preserved.

Each section of a sheaf over a point returns a single record in the fiber. The sheaf over an open set U surrounding a point k is called a *stalk* [66], [67]

$$\mathcal{O}_{K,E} \restriction_k := \lim_{U \ni k} \Gamma(U, E \restriction_U) \tag{9}$$

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 and the germ include 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.

3.3 Data Index and Graphic Index Correspondence

There is an expectation that for a visualization to be readable, the visual elements must correspond to distinct data elements [21] and we can use the properties of sheaves to formally express this correspondence. We first describe the relationship between the graphic indexing space S and the data indexing space K which we propose is one where multiple graphic indexes map to one data index, and every index in the graphic space can be mapped to an index in the data space. We encode these expectations as the map ξ, which we define to be a surjective continuous map

$$\xi : W \to U \tag{10}$$

between a graphic subspace $W \subseteq S$ and data subspace $U \subseteq K$. The functor ξ is surjective such that for every point $k \in U$ there is a corresponding set of points $\{s | s \in \xi^{-1}(k)\}$ for all $s \in W$.

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. As exemplified in Figure 5, we define ξ as a surjective map because it allows us to express that a union of graphic spaces S_i maps to single data point k, which allows us to express visual representations of a single record that are the union of many primatives, discussed in Equation 33, such as multipart glyphs (e.g boxplots) and combinations of plot types (e.g line with point markers).

3.3.1 *Data and Graphic Correspondence*

Since we have defined a function ξ between two spaces K, S, we can then construct functors that transport sheaves over each space to the other [67]. This allows us to describe what data we expect at each graphic index location and what graphic is expected at each data index location. Transport functors compose the indexing map ξ with the sheave map to say that a record τ at k is at all corrresponding s and that a function ρ over one point s is the same function at all points $s \in S$ that correspond to the same record index k.

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

Definition 3.9. Given a sheaf $\mathcal{O}_{S,H}$ on S, the **pushforward** sheaf $\xi_*\mathcal{O}_{S,H}$ on K is defined as

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

for all opensets $U \subset K$ [67].

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 ξ_* transports sheaves of sections on W over U

$$\Gamma(U, \xi_*H \restriction_U) \ni \xi_*\rho : U \to \xi_*H \restriction_U \tag{11}$$

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

$$\xi_*\rho(k) = \rho \restriction_{\xi^{-1}(k)} \tag{12}$$

such that $\xi_*\rho(k))(s) = \rho(s)$ for all $s \in \xi^{-1}(k)$. Therefore, the continuous map ξ and transport functors ξ^*, ξ_* allow us to express the correspondence between graphic section and data section.

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

Definition 3.10. [67] 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 ξ^* transports sheaves of sections on $U \subseteq K$ over $W \subseteq S$

$$\Gamma(W, \xi^*E \restriction_W) \ni \xi^*\tau : W \to \xi^*E \restriction_W \tag{13}$$

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

$$\xi^*\tau(s) = \tau(\xi(s)) = \tau(k) \tag{14}$$

As ξ is surjective, there are many points $s \in W \subseteq S$ in the graphic space that correspond to a single point $\xi(s) = k$.

3.3.2 *Example: Graphic and Data*

Functors between sheaves are a way of expressing the bookkeeping involved in keeping track of which graphic section ρ corresponds to which data section τ. The (k_i, S_i) pairing expressed in Equation 10 establishes that there is a correspondence between sections evaluated over k_i and S_i. This allows us to construct graphic specifications for each data index $\xi_*\rho$ and retrieve the data $\xi^*\tau$ for any graphic section generating any piece of a graphic. In Figure 5, the visualization is a graphic representation of a unit circle and the sin and cosine curves on that interval. The index lookup ξ describes which parts of the circle and curves

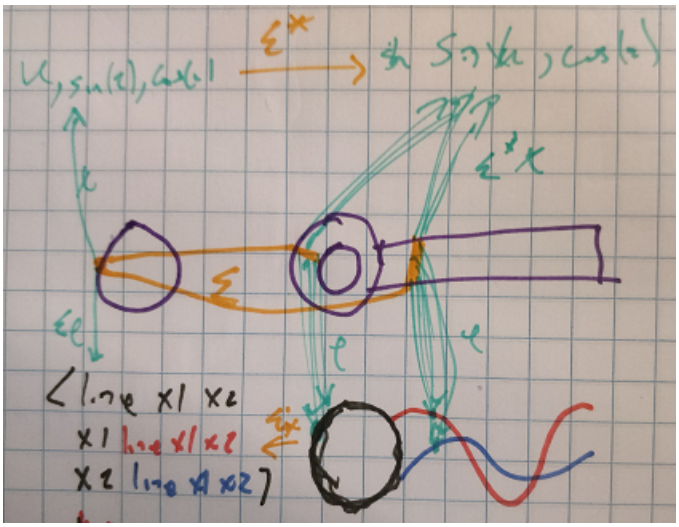


Fig. 5. The data consists of the $\sin$ and $\cos$ functions over a unit circle base space. We choose to visualize this as a circle and two line plots. The indexing function ξ bookkeeps which parts of the circle and each curve correspond to each point on the unit circle. The pushforward ξ_* matches each point in the data space to the specification of the graphic at that point, while the pullback ξ^* matches each point in the graphic space to the data over that point.

are generated from which points on the unit circle. Given this correspondance, the pullback ξ^* looks up which values are being represented in a given part of the graphic. The pushforward ξ_* describes how a graphic is supposed to look for each point in the data space. The graphic parameterized in Figure 5 is intended as an approximation of $\xi_*\rho$.

Visualization specification languages such as vega [68] and svg [69] assume that there is a graphic specification for every point. Interactive tooltips assume that there is a map between the graphics rendered on screen and the data such that a glyph maps back to a record. The expectation that these mappings between data and graphic space exist are encapsulated in the functors (ξ, ξ_*, ξ^*).

4 Artist: Data to Graphic

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:

$$A : \Gamma(K, E) \to \Gamma(S, H) \tag{15}$$

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 4.1. 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 α_c. [70]

This means that natural transforms are maps of functors that take the same input object and return objects in the same category [71]. As illustrated in Equation 53, the sheaf functors

$$\Gamma(K, E) \xleftarrow{\mathcal{O}_{K,E}} K \xmapsto{\xi_* \mathcal{O}_{S,H}} \Gamma(K, \xi_* H) \tag{16}$$

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 ι, ι^* morphisms of the presheaf functor, described in 3.2 and 3.2, such that the following diagram commutes:

$$\begin{array}{ccc} K_1 & \Gamma(K_1, E) & \xrightarrow{A_{K_1}} \Gamma(K_1, \xi_* H) \\ \uparrow \iota & \downarrow \iota^* & \downarrow \iota^* \\ K_2 & \Gamma(K_2, E) & \xrightarrow{A_{K_2}} \Gamma(K_2, \xi_* H) \end{array} \tag{17}$$

The diagram in Equation 17 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 tra nslate 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 [72] and panning and zooming [73]

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

$$\mathrm{Im}_A(S, H) \coloneqq \{\rho \mid \exists\, \tau \in \Gamma(K, E) \text{ s.t. } A(\tau) = \rho,\ \xi(S) = K\} \tag{18}$$

that are, by definition, only reachable through a structure preserving artist, which we describe in subsubsection 4.1.2. 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.

4.1 Equivariance

As introduced in subsection 2.2, 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 [74] 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).

4.1.1 *Mathematical Structure of Data*

We separate data transformations into two components, transformations on the base space $(\hat{\phi}, \hat{\phi}^*)$ and transformations on the fiber space $\tilde{\phi}$.

$$\begin{array}{ccc} \Gamma(U, E \restriction_U) & \xmapsto{\hat{\phi}^*} & \Gamma(U', \hat{\phi}^* E \restriction_{U'}) \\ \uparrow & & \uparrow \\ U & \xleftarrow{\hat{\phi}} & U' \end{array} \qquad \begin{array}{c} \Gamma(U', \hat{\phi}^* E \restriction_{U'}) \\ \downarrow \tilde{\phi} \\ \Gamma(U', \hat{\phi}^* E \restriction_{U'}) \end{array} \tag{19}$$

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 $\hat{\phi}^* \tau$.

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

$$\hat{\phi} : U' \to U \tag{20}$$

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

$$\hat{\phi}^* \tau \restriction_{U'} : \tau \restriction_{U'} \mapsto \tau \restriction_{U' \circ \hat{\phi}} \tag{21}$$

such that $\tau|_U = \hat{\phi}^* \tau|_{U'}$ because $\tau|_U = \tau|_{\hat{\phi}(U')}$. This means that the base space transformation $\hat{\phi}$ does not change the data values at a given point

$$\tau(K) = \hat{\phi}^* \tau(k') = \tau(\hat{\phi}(k')) \tag{22}$$

where $\hat{\phi}(k') = \hat{\phi}(k)$.

4.1.1.2 **Records**: As introduced in Equation 19, the fiber transformation $\tilde{\phi} : \hat{\phi}^* E_{k'} \to \hat{\phi}^* E_{k'}$ is a monoid action on the fiber $\tilde{\phi} \in \mathrm{Hom}(\hat{\phi}^* F \restriction_{k'}, \hat{\phi}^* F \restriction_{k'})$ restricted to a point $k' \in U'$. The fiber transformation is a change in section

$$\tilde{\phi} : \hat{\phi}^* \tau \restriction_{U'} \mapsto \hat{\phi}^* \tau' \restriction_U \tag{23}$$

where $\tau, \tau' \in \Gamma(U', \hat{\phi}^* E \restriction_{U'})$. Since there are many different fiber transformations, there is not a uniform way of expressing how this morphism acts on sections unlike in Equation 22. Examples of a fiber transformation are actions such as partial ordering for ranking purpose [75] and the permutation, ordering, translation, and scaling codified as Steven's measurement scales [12], [13], [76].

4.1.1.3 **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

$$\phi : \tau \restriction_U \mapsto \tau' \restriction_U \circ \hat{\phi} \tag{24}$$

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

The data transform ϕ is composable

$$\phi = (\hat{\phi}, \prod_{i=0}^{n} \tilde{\phi}_i) \tag{25}$$

if each (identical) component base space is transformed in the same way $\hat{\phi}$ and there exists functions $\phi_{a,b} : E_a \times E_b \to E_a \times E_b$, $\phi_a : E_a \to E_a$ and $\phi_b : E_b \to E_b$ 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.

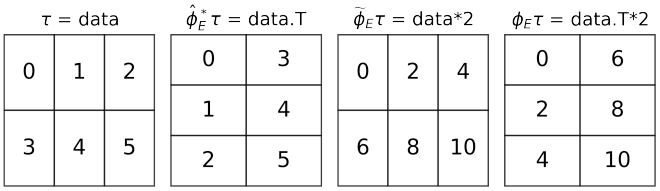
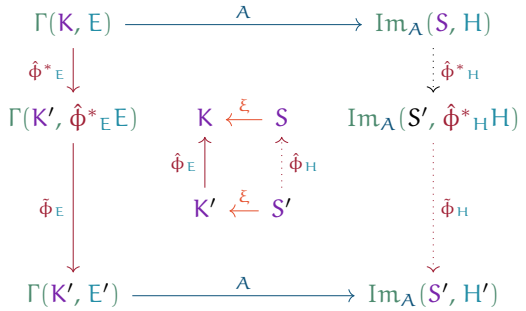


Fig. 6. Values in a data set can be transformed in three ways: $\hat{\phi}$-values can change position, .e.g transposed; $\tilde{\phi}$-values can change, e.g. doubled; ϕ - values can change position and value

Figure 6 provides an example of a transposition base space change $\hat{\phi}$, a scaling fiber space change $\tilde{\phi}$, and a composition of the two ϕ 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 τ because ϕ does not change the continuity. Each value in $\hat{\phi}^*\tau$ is also the same as in τ, just moved to the new position. In $\tilde{\phi}\tau$, each value is scaled by two but remains in the same location as in τ. And in $\phi\tau$ each function is transposed such that it retains its neighbors and all values are scaled consistently.

4.1.2 *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 $s' \in S'$



(26)

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

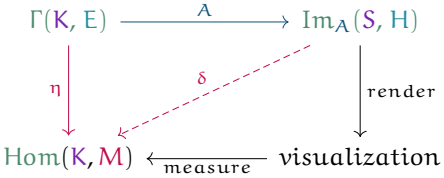
$$A(\tilde{\phi}_E(\tau(\hat{\phi}_E(\xi(\hat{\phi}_H(s')))))) = \tilde{\phi}_H(A(\tau(\xi(\hat{\phi}_H(s'))))))$$

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 δ 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.

$$\delta : (\rho \circ \xi^{-1}) \mapsto (K \xrightarrow{\delta_\rho} M) \tag{27}$$

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.



(28)

The extraction function is equivalent to measuring components of the rendered image $\delta = \texttt{measure} \circ \texttt{render}$, 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 η that maps data to the measurement space directly

$$\eta : \tau \mapsto (K \xrightarrow{\eta_\tau} M) \tag{29}$$

such that $\eta_\tau(k)$ is the expected set of measurements M_k. The pair of verification functions (η, δ) can be used to test that the expected encoding η_τ of the data matches the actual encoding δ_ρ

$$\eta(\tau)(k) = \delta(A(\tau))(k) = \delta(\rho \circ \xi^{-1})(k) = M_k \tag{30}$$

An artist is equivariant when changes to the input and output are equivariant.

As introduced in Equation 20, 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

$$\eta(\tau)(\hat{\phi}(k')) = \delta(A(\tau))(k) \tag{31}$$

On the other hand, a change in sections Equation 23 induces an equivalent change in measurements

$$\eta(\tilde{\phi}(\tau))(k) = \tilde{\phi}_M(\delta(A(\tau))(k)) \tag{32}$$

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.

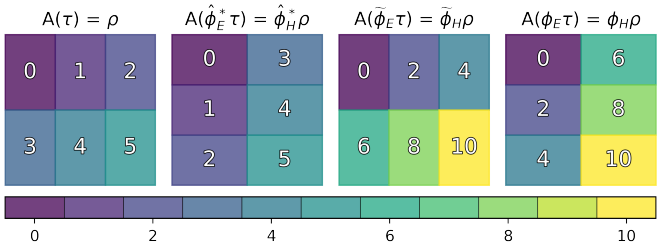
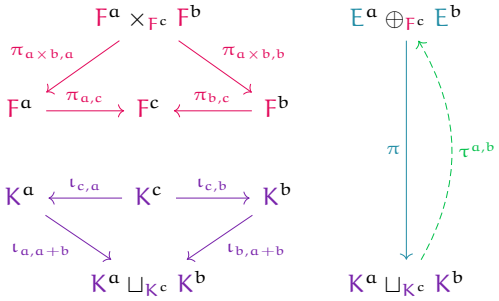


Fig. 7. This artist is equivariant because when the input data τ is transposed, $\hat{\phi}$, scaled $\tilde{\phi}$, and transposed and scaled ϕ, the corresponding colored cells are transposed, scaled such that the color is moved two steps, and both transposed and scaled.

For example, in Figure 7, the measurable variable is color. This is a visual representation of the data shown in Figure 6, 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 31, 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 32, 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 ϕ 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 ϕ panel.

4.2 Composing Artists

A common use of category theory in software engineering is the specification of modular components [22] 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.



(33)

4.2.1 Addition

As illustrated in Equation 33, data bundles can be combined by taking the disjoint union of base spaces $K^a \sqcup_{K^c} K^b$, where K^c is an overlap. When the fibers of each bundle are isomorphic $F^a \simeq F^b$, which we denote as E, this is analogous to adding more records to a dataset. 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 Equation 15, the artist returns a function ρ. 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

$$\rho(s) \coloneqq \begin{cases} \rho^a(s) & s \in \xi^{-1}(K^a) \\ \rho^b(s) & s \in \xi^{-1}(K^b) \end{cases} \tag{34}$$

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

$$\begin{aligned} &A_{a+b}(\tau^{a+b}(k_a)) \\ &= A_a(\tau^a(k_a)) = A_b(\tau^b(k_b)) \end{aligned} \tag{35}$$

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 ρ_4. If that point is also in the 3rd quadrant $k \in K_3$, then both artist outputs must return the same values $\rho_4(\xi^{-1}(k)) = \rho_3(\xi^{-1}(k))$.

4.2.2 Multiplication

As illustrated in Equation 33, fibers that are a cartesian product of fiberspaces $F^a \times_{F^c} F^b$, where F^c is any fiber that is present in both fibers, can be projected down into component fibers. 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

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

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

$$\rho^{a\times b} = \{\rho^a(s), \rho^b(s)\}, s \in \xi^{-1}(K) \tag{36}$$

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 4.2.1, 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.

$$\begin{aligned} &\delta(A_{a\times b}(\tau^{a\times b}(k))) \restriction_{F^c} \\ &= \delta(A_a(\tau^a(k_a))) \restriction_{F^c} = \delta(A_b(\tau^b(k_b))) \restriction_{F^c} \end{aligned} \tag{37}$$

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 Zening and Hullman [61]. This expectation can also be used to check that a multipart glyph is assembled correctly. For example, a box plot [77] 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.

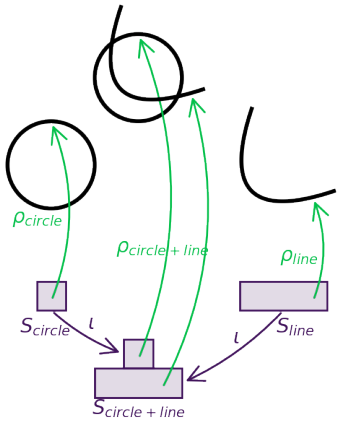


Fig. 8. The circle-line visual element can be constructed via $\rho_{circle} + \rho_{line}$ functions that generate the circle and line elements respectively. This is equivalent to a $\rho_{circle+line}$ function that takes as input the combined base space $S_{circle} \sqcup S_{line} = S_{circle-line}$ 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 8 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:

$$A(\Gamma(\bigsqcup_i K^i, \bigoplus_i E^i)) \coloneqq \sum_i A_i(\Gamma(K^i, E^i)) \tag{38}$$

As shown in Equation 33, 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.

5 Construction

We propose that one way of constructing artist functions is to separate generating a visualization into an encoding stage ν and a compositing stage Q. In the encoding stage ν, a data bundle is treated as separable fields and each field is mapped to a measurable visual variable. In the encoding stage, the expected visual mappings η 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.

5.1 Measurable Visual Components

We propose an intermediate visual fiber bundle

$$P \hookrightarrow V \xrightarrow{\pi} K \tag{39}$$

where the space of possible visual encodings is the fiber space P. The space of visual sections which return visual encoding specifications is

$$\Gamma(U, V\restriction_U) \coloneqq \{\mu : U \to V\restriction_U \mid \pi(\mu(k)) = k \text{ for all } k \in U\} \tag{40}$$

As shown in Equation 28, measurable visual components are defined as having the same continuity as the data K. This means that every data record $\tau(k)$ has a corresponding visual section such that $\pi(\tau(k)) = \pi(\mu(k))$.

Although the bundle V is structurally equivalent to the bundle E, its existence allows for separating the data that is input into the artist τ from the data internal to the artist μ. This allows a visualization library to define a visual bundle space V the holds the internal representation standard for measurable visual components. For example, a visualization library could define a visual color fiber as an RGBA tuple $P_{color} = \mathbb{R}^3 \times [0,1]$. This would then set the expectation that arbitrary color encoding functions would need to return an RGBA tuple for the library to recognize the encoding as a color.

5.2 Map Between Graphics and Data

In subsection 3.2, the functors ξ, ξ^*, ξ_* are introduced to describe the expected relationship between screen and data base spaces. In the construction of the artist, the functor ξ is defined such that the data base space K is a deformation retraction [48], [78] of the graphic space S. This means that there is a continuous surjective mapping from every point $k \in K$ to a point $k \in dbase$. To simplify matters, in this paper, we construct the graphic space as a constant multiple of the base space such that

$$\underbrace{K \times [0,1]^n}_{S} \xmapsto{\xi} K \tag{41}$$

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

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

because the data dimensionality K may be too small for a graphic representation.

Fig. 9. The graphic base space S is collapsable 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 9, 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.

5.3 Component Encoders

The encoding function ν is a map from data sections to graphic sections

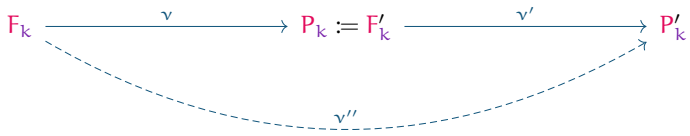
$$\nu : \Gamma(K, E) \to \Gamma(K, V) \tag{42}$$

such that the sections project to the same point on the base space $\pi(E) = \pi(\nu(E))$. A consequence of this property is that ν can be constructed as a pointwise transformation such that

$$\nu : F_k \to P_k \tag{43}$$

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 ν can convert a single record and may not need the whole dataset.

The difference between E and V are semantic rather than structural; they are both maps from the topological space K to sets of functions that return records of values. This means any V can be redefined as E,

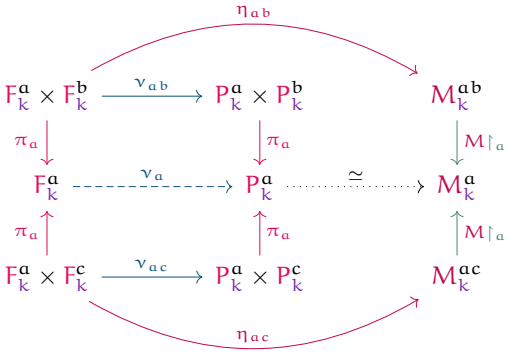


(44)

which means that, as shown in Equation 44, any collection of ν functions can be composed such that they are equivalent to a ν that directly converts the input to the output. As with artists, ν are maps of sections such that the operators defined in subsection 4.2 can also act on transformers ν, 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.

5.3.1 *Encoder Verification*

A motivation for constructing an artist with an encoder stage ν is so that the conversion from data to measurable component can be tested separately from the assembly of components into a glyph.



(45)

As shown in Equation 45, an encoder is considered valid if there is an ismorphism 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 ν is equivariant if the change in data, as defined in subsubsection 4.1.1, and change in visual components are equivariant. Since E and V are over the same base space and are pointwise, the base space change $\hat{\phi}_E$ applies to both sides of the equation

$$\nu(\tau_E(\hat{\phi}_K(k'))) = \mu(\hat{\phi}_K(k')) \tag{46}$$

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

$$\tilde{\phi}_V \nu(\tau(k)) = \nu(\tilde{\phi}_E(\tau(k))) \tag{47}$$

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 2.2, this is why Bertin and many others since have advocated choosing an encoding that has a structure that matches the data structure [6]. For example choosing a quantative colormap to encode quantative data if the $\tilde{\phi}$ operation is scaling, as in Figure 7.

5.4 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 μ into sections of graphics ρ.

$$Q : \Gamma(K, V) \to \Gamma(S, H) \tag{48}$$

The compositing function is map from sheaves over K to sheaves over S. This is because, as described in Figure 9, 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 44, a new compositor function Q can be constructed by precomposing ν functions with the existing Q.

$$\Gamma(K, V) \xrightarrow{\nu} \Gamma(K, V') \xrightarrow{Q} \Gamma(S, H) \qquad (49)$$
$$\Gamma(K, V) \xrightarrow{Q'} \Gamma(S, H)$$

The composition in Equation 49 means that different measurable components can yield the same visual elements. The operators defined in subsection 4.2 can also act on compositors Q such that $Q_{a+b} = Q_a + Q_b$ and multiplied d $Q_{a\times b} = Q_a Q_b$.

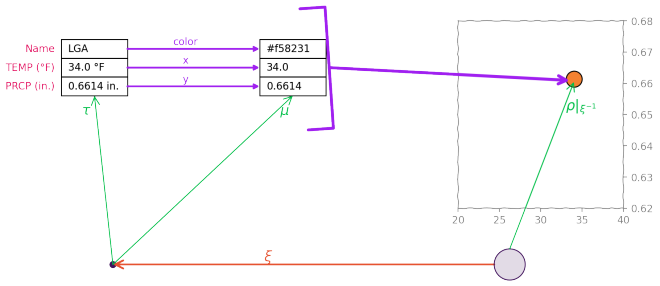


Fig. 10. This simple Q assembles a circular visual element that is the color specified in $\mu(k)$ and is at the intersection specified in $\mu(k)$

As shown in Figure 10, a set of ν 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 ρ. 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 μ. The composition rule in Equation 49 means that developers can implement Q as drawing circles or can implement a Q that draws arbitrary shapes, and then provide different ν adapters, such as one that specifies that the shape is a circle.

5.4.1 *Compositor Verification*

An advantage of factoring out encoding and verification, as discussed in subsubsection 5.3.1, is that the responsibility of the compositor can be scoped to translating measurable components into visual elements.

$$\begin{array}{ccc} \Gamma(K, V^a \times V^b) & \xrightarrow{Q_{ab}} & \mathrm{Im}_{A_{ab}}(S, H) \\ \downarrow{\scriptstyle \pi_a} & & \downarrow{\scriptstyle M\restriction_a \circ \delta_{ab}} \\ \Gamma(K, V^a) & \xrightarrow{\simeq} & \mathrm{Hom}(K, M^a) \\ \uparrow{\scriptstyle \pi_a} & & \uparrow{\scriptstyle M\restriction_a \circ \delta_{ac}} \\ \Gamma(K, V^a \times V^c) & \xrightarrow{Q_{ac}} & \mathrm{Im}_{A_{ac}}(S, H) \end{array} \qquad (50)$$

As illustrated in Equation 50, 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 μ_{ab} and μ_{ac}.

A compositor function Q is equivariant if the renderer output changes in a way equivariant to the data transformation defined in subsubsection 4.1.1. 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

$$\mu(\hat{\phi}_K(k')) = \delta(Q(\mu)(\hat{\phi}_K(\xi^{-1k}))) = M_k \qquad (51)$$

As discussed in Figure 7, 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

$$\tilde{\phi}_V(\mu(k)) = \tilde{\phi}_M(Q(\mu(k))) \qquad (52)$$

which means that any change to a measurable component input must have a measurably equivalent change in the output. As illustrated in Figure 7, 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.

5.5 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 [67] 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.

$$\begin{array}{ccc} \Gamma(U, E\restriction_U) & \xrightarrow{\xi^*} & \Gamma(W, \xi^* E\restriction_W) \\ {\scriptstyle \mathrm{Hom}_{\mathcal{O}_K}}\downarrow & \searrow^{\mathrm{Hom}_{\mathcal{O}_K, \mathcal{O}_S}} & \downarrow{\scriptstyle \mathrm{Hom}_{\mathcal{O}_S}} \\ \Gamma(U, \xi_* H\restriction_U) & \xleftarrow{\xi_*} & \Gamma(W, H\restriction_W) \end{array} \qquad (53)$$

Since the artist is a family of functions in the homeset between sheaves, the isomorphism allows for the specification of the transformation from data as combination

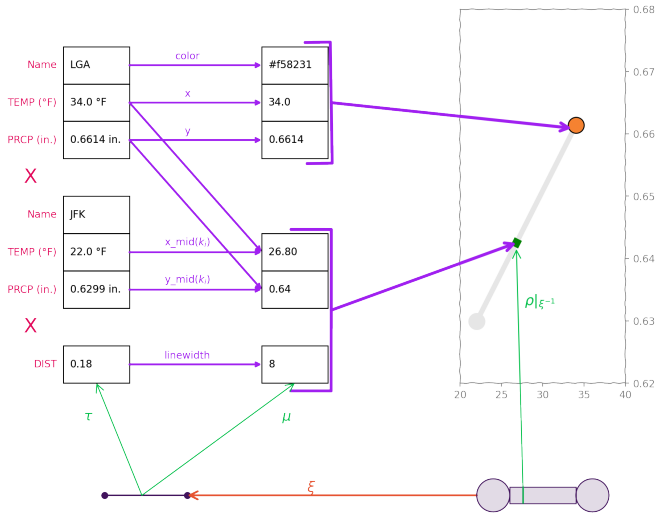
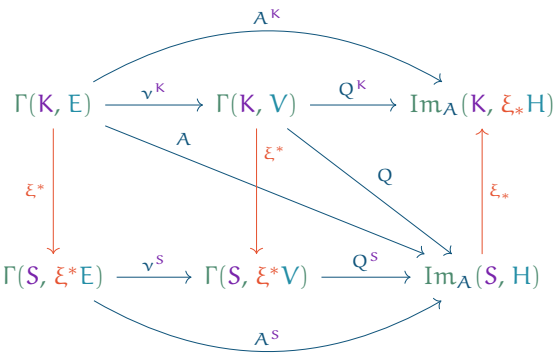


Fig. 11.

of functions over different spaces such that the following diagram commutes:



(54)

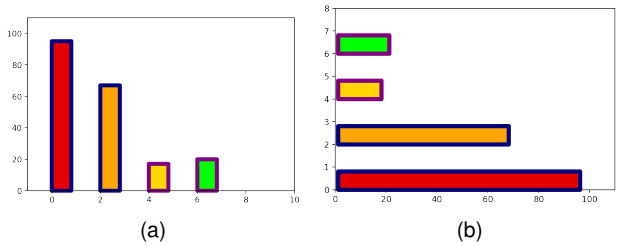
This means that an artist over data space $A_K : \tau \mapsto \xi_* \rho$, an artist over graphic space $vartist_S : \xi^* \tau \mapsto \rho$, and an artist $A : \tau \mapsto \rho$ are equivalent such that:

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

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

6 Discussion: Feasibility as Design Spec

The framework specified in section 4 and section 5 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 5 to illustrate how components that adhere to these specifications are maintainable, extendible, scalable, and support concurrency.



Specially, we introduce artists for building the graphical elements shown in 6 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] [30], [79] to show that components using this model can be incorporated into existing visualization libraries iteratively. While the architecture specified in section 5 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.

6.1 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 $F = \{\texttt{apple}, \texttt{orange}, \texttt{lemon}\} \times \mathbb{Z}^+ \times \{\texttt{True}, \texttt{False}\}$. We thinly wrap subsection 6.1 in an object so that the common data interface function is that $\tau = \texttt{DataContainerObject.query}$.

```
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 [80], pandas [81], xarray [82], and networkx [42]. 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 treat the data as a

[4]Matplotlib artists are our artist's namesake

collection of single fiber bundles. This is equivalent to the total bundle, as shown in Equation 33.

6.2 Component Encoders

To encode the values in the dataset, we enforce equivariance by writing ν 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 ν is a python dictionary that returns an integer position, because Matplotlib's internal parameter space expects a numerical position type.

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

As mentioned in Equation 44, the encoders can be composed up. For example, the compositor ν may need the position to be converted to screen coordinates. Here the screen coordinate ν 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.

```
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` $\nu_{\texttt{transData}}$ composed with the position encoder $\nu_{\texttt{position}}$ 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 ν encoders that are explicit about which ϕ_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 ν encoders that are appropriate for their data.

6.3 Graphic Compositors

After converting each record into an intermediate visual component μ, the set of visual records is passed into Q. Here the Q includes one last encoder, as illustrated in Equation 49, that assembles the independent visual components into a rectangle. This ν 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 54. This means that the parameters are constructed in data space K and this function returns a pushed forward $\xi_*\rho$.

```
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 28, the renderer is supposed to take ρ as input such that $\mathit{renderer}(\rho) = \mathit{visualization}$, 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 12a and 12b. In this model, the horizontal bar is implemented as a composition of a ν that renames fields in $\mu_{\texttt{barh}}$ and the Q implementation for the horizontal bar.

```
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 $Q_{\texttt{barh}} = Q_{\texttt{bar}} \circ \nu_{\texttt{vtoh}}$, which is an example of Equation 49. These functions can be further added together, as described in subsection 4.2 to build more complex visualizations.

6.4 Integrating Components into an Existing Library

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

```
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
```

This artist is then passed along to a shim artist that makes it compatible with existing Matplotlib objects

```
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)
```

As shown in the `draw` method, generating a graphic section ρ 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 ν 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.

The `GenericArtist` is a a standard Matplotlib object; therefore it can be hooked into the Matplotlib draw tree to produce the vertical bar chart in 12a. 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. 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.

7 Conclusion

The toy example presented in section 6 demonstrates that it is relatively straightforward to build working visualization library components using the construction described in section 5. Since these components are defined with single record inputs, the 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 location/subset/indicies	fiber space fields	total space dataset type
Data	$k \in U \subseteq K$	F	E
Visual	$k \in U \subseteq K$	P	V
Graphic	$s \in W \subseteq S$	D	H

TABLE 1
Topological spaces introduced in subsection 3.1

	section record at location	sheaf 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 2
Functions that associate topological subspaces with records, discussed in subsubsection 3.1.3 and subsection 3.2

	function	constraint
s to k	$\xi : W \to U$	for $k \in U$ exists $s \in W$ 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 3
Functors between graphic and data base spaces subsection 3.3

changing locations, moving records to new locations, changing records
TABLE 4
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 distinguised through subscripts ϕ_E, ϕ_V and ϕ_H

color

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

TABLE 5
artist, verification functions, and construction $A = Q \circ \nu$ introduced in section 4, and section 5

These functions can be verified using the following functions:

Acknowledgments

The authors would like to thank...

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.

[2] J. Hughes, "Why Functional Programming Matters," *The Computer Journal*, vol. 32, no. 2, pp. 98–107, Jan. 1989.

[3] D. A. Norman, *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[4] E. R. Tufte, *The Visual Display of Quantitative Information*. Cheshire, Conn.: Graphics Press, 2001.

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

[6] J. Bertin, *Semiology of Graphics : Diagrams, Networks, Maps*. Redlands, Calif.: ESRI Press, 2011.

[7] J. H. Lawrimore, M. J. Menne, B. E. Gleason, C. N. Williams, D. B. Wuertz, R. S. Vose, and J. Rennie, "Global Historical Climatology Network - Monthly (GHCN-M), Version 3." 2011.

[8] 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.

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

[10] 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.

[11] 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.

[12] W. A. Lea, "A formalization of measurement scale forms," p. 44.

[13] 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.

[14] nLab authors, "Action," Jul. 2022.

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

[16] 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.

[17] nLab authors, "Equivariant," Jul. 2022.

[18] A. M. Pitts, *Nominal Sets: Names and Symmetry in Computer Science*. USA: Cambridge University Press, 2013.

[19] J. Mackinlay, "Automatic Design of Graphical Presentations," Ph.D. dissertation, Stanford, 1987.

[20] 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.

[21] 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.

[22] 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.

[23] B. A. Yorgey, "Monoids: Theme and Variations (Functional Pearl)." New York, NY, USA: Association for Computing Machinery, p. 12.

[24] 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.

[25] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, vol. 5, no. 2, pp. 110–141, Apr. 1986.

[26] 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.

[27] 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.

[28] 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.

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

[30] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, May 2007.

[31] 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.

[32] 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.

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

[34] 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.

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

[36] 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.

[37] J. VanderPlas, B. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert, "Altair: Interactive Statistical Visualizations for Python," *Journal of Open Source Software*, vol. 3, no. 32, p. 1057, Dec. 2018.

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

[39] 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.

[40] S. Studies, "Culturevis/imageplot," Jan. 2021.

[41] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An Open Source Software for Exploring and Manipulating Networks," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 3, no. 1, Mar. 2009.

[42] 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.

[43] D. I. Spivak, "Databases are categories," Jun. 2010.

[44] ——, "Simplicial databases," 2009.

[45] "Fiber bundle," *Wikipedia*, May 2020.

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

[47] "Restriction (mathematics)," *Wikipedia*, Jun. 2022.

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

[49] J. R. Munkres, *Elements of Algebraic Topology*. Menlo Park, Calif: Addison-Wesley, 1984.

[50] T.-D. Bradley, J. Terilla, and T. Bryson, *Topology : A Categorical Approach*, 2020.

[51] E. W. Weisstein, "Open Set," https://mathworld.wolfram.com/.

[52] T.-D. Bradley, "Topology vs. "A Topology" (cont.)."

[53] "Quotient space (topology)," *Wikipedia*, Nov. 2020.

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

[55] E. Riehl, *Category Theory in Context*.

[56] S. Mac Lane, *Categories for the Working Mathematician*. Springer Science & Business Media, 2013, vol. 5.

[57] B. Fong and D. I. Spivak, *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*, 1st ed. Cambridge University Press, Jul. 2019.

[58] J. D. Ullman and J. Widom, *A First Course in Database Systems*. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

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

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

[61] 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.
[62] "Cairographics.org."
[63] T.-D. Bradley, "What is a Functor? Definitions and Examples, Part 2," https://www.math3ma.com/blog/what-is-a-functor-part-2.
[64] nLab authors, "Presheaf," Oct. 2021.
[65] M. J. Baker, "EuclideanSpace: Maths - Sheaf," https://www.euclideanspace.com/maths/topology/sheaf/.
[66] "Stalk (sheaf)," *Wikipedia*, Oct. 2019.
[67] G. Harder and K. Diederich, *Lectures on Algebraic Geometry I: Sheaves, Cohomology of Sheaves, and Applications to Riemann Surfaces*, ser. Aspects of Mathematics. Vieweg+Teubner Verlag, 2008.
[68] J. Heer and M. Bostock, "Declarative language design for interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1149–1156, Nov. 2010.
[69] A. Quint, "Scalable vector graphics," *IEEE MultiMedia*, vol. 10, no. 3, pp. 99–102, Jul. 2003.
[70] T.-D. Bradley, "What is a Natural Transformation? Definition and Examples," https://www.math3ma.com/blog/what-is-a-natural-transformation.
[71] B. Milewski, "Category Theory for Programmers," p. 498.
[72] 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.
[73] 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.
[74] J. Bertin, "II. The Properties of the graphic system," in *Semiology of Graphics*. Redlands, Calif.: ESRI Press, 2011.
[75] R. Brüggemann and G. P. Patil, *Ranking and Prioritization for Multi-indicator Systems: Introduction to Partial Order Applications*. Springer Science & Business Media, Jul. 2011.
[76] S. S. Stevens, "On the Theory of Scales of Measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946.
[77] H. Wickham and L. Stryjewski, "40 years of boxplots," *The American Statistician*, 2011.
[78] nLab authors, "Deformation retract," Dec. 2021.
[79] J. Hunter and M. Droettboom, "The Architecture of Open Source Applications (Volume 2): Matplotlib," https://www.aosabook.org/en/matplotlib.html.
[80] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
[81] J. Reback, W. McKinney, jbrockmendel, J. V. den Bossche, T. Augspurger, P. Cloud, gfyoung, Sinhrks, A. Klein, M. Roeschke, S. Hawkins, J. Tratner, C. She, W. Ayd, T. Petersen, M. Garcia, J. Schendel, A. Hayden, MomIsBestFriend, V. Jancauskas, P. Battiston, S. Seabold, chris-b1, h-vetinari, S. Hoyer, W. Overmeire, alimcmaster1, K. Dong, C. Whelan, and M. Mehyar, "Pandas-dev/pandas: Pandas 1.0.3," Zenodo, Mar. 2020.
[82] 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.

Michael Grossberg Biography text here.