Make any stupid plot you want

Hannah Aizenman

A dissertation proposal submitted to
the Graduate Faculty in Computer Science in partial fulfillment of the
requirements for the degree of Doctor of Philosophy,
The City University of New York

Committee Members:
Dr. Michael Grossberg (Advisor), Dr. Robert Haralick, Dr. Lev Manovich,
Dr. Huy Vo

Abstract

A critical aspect of data visualization is that the graphical representation of data match the properties of the data; this fails when order is not preserved in representations of ordinal data or scale for numerical data. In this work, we propose that the mathematical notion of equivariance formalizes the expectation that graphics match the data. We developed a model we call the topological artist model (TAM) in which data and graphics can be viewed as sections of fiber bundles. This model allows for (1) decomposing the translation of data fields (variables) into visual channels via an equivariant map on the fibers and (2) a topology-preserving map of the base spaces that translates the dataset connectivity into graphical elements. Furthermore, our model supports an algebraic sum operation such that more complex visualizations can be built from simple ones. We illustrate the application of the model through case studies of a scatter plot, line plot, and heatmap. We show that this model can be implemented with a small prototype.

To demonstrate the practical value of our model, we propose a model driven re-architecture of the artist layer of the Python visualization library Matplotlib. We represent the topological base spaces using triangulation, make use of programming types for the fiber, and build on Matplotlib's existing infrastructure for the rendering. In addition to providing a way to ensure the library preserves structure, the functional decomposition of the artist in the model could improve modularity, maintainability, and point to ways in which the library could better support concurrency and interactivity. The thesis will follow through on this proposal to explore how to further develop our model, showing how it can support Matplotlib's current diverse range of data visualizations while providing a better platform for domain-specific visualization library developers.

Contents

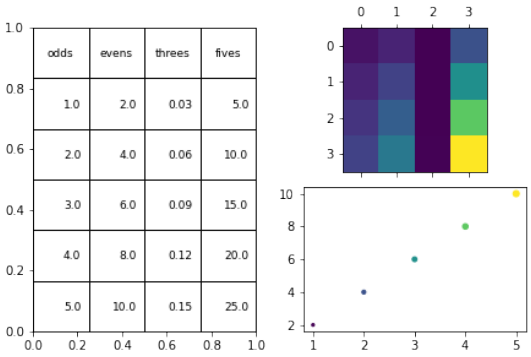


Figure 1: Implicit in visualization is the assumption that these three representations of data are equivalent, specifically that the measurements within a variable and relations of the measurements of each variable are preserved.

1 Introduction

1.1 Thesis statement

We define a visualization as a transform from data to graphic that preserves the topology of the data and faithfully map the properties of the measurement type. In fig 1, we implicitly assume that the translation from table to heatmap has preserved the order of observations (the rows) and that the perceptually uniform sequential colormap has been applied such that the ordering relation on floats matches the ordering on the colormap (darker colors map to larger numbers). We also make this assumption about color in the scatter map, and that the translation to size and position on screen also respect the ordering on floats. In this work, we propose to mathematically describe the transform of data to visual space such that we can make explicit the implicit topology and types visualizations preserve. We then propose a new architecture for the Python visualization library Matplotlib [13] based on these descriptions because the Matplolib artist layer is analogous to the transforms.

1.2 what are we doing

constraints: toplogy of dataset + structure on the variables implementation: be fully feature we can haz compose the things into larger structure. & has framwework to do larger things w/ algebraic formalizations.

1.3 What are the the basic requirements of a visualization library?

Acquired codes of meaning, dubois

2 Not all data are tables

Tables, images (Lev), graphs (network X)

set up: dubois

theorists: bertin, munzner, mackinlay

talk about: matplotlib arch paper, excel/matlab arch, vtk & ggplot (compare/contrast, we're blending these things)

2.1 Terminology

Is currently C&P'ed from my second exam so needs some messaging, but

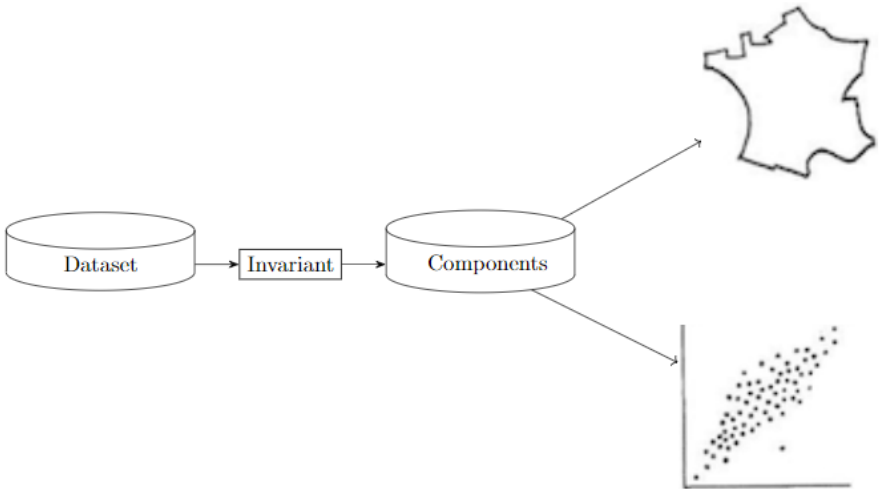


Figure 2: To go from a dataset to a visualization, the data is subset based on a set of constraints (the invariant). The resulting subset becomes the components that are visualized, but the choice of visualization is dependent on the type and structure of the component variables.

Given a dataset, we need to decide what subset of the data to visualize. Bertin describes the set of constraints used to subset the data as the *invariant*. Formally, the *invariant* is the

set of shared characteristics of the data being visualized. When these constraints are applied to the dataset, the resulting subset is what will become the *components* of the visualization [4]. As shown in figure 2, the final step in creating a visualization is choosing how to encode the components using retinal (visual) variables.

2.1.1 Retinal (Visual) Variables

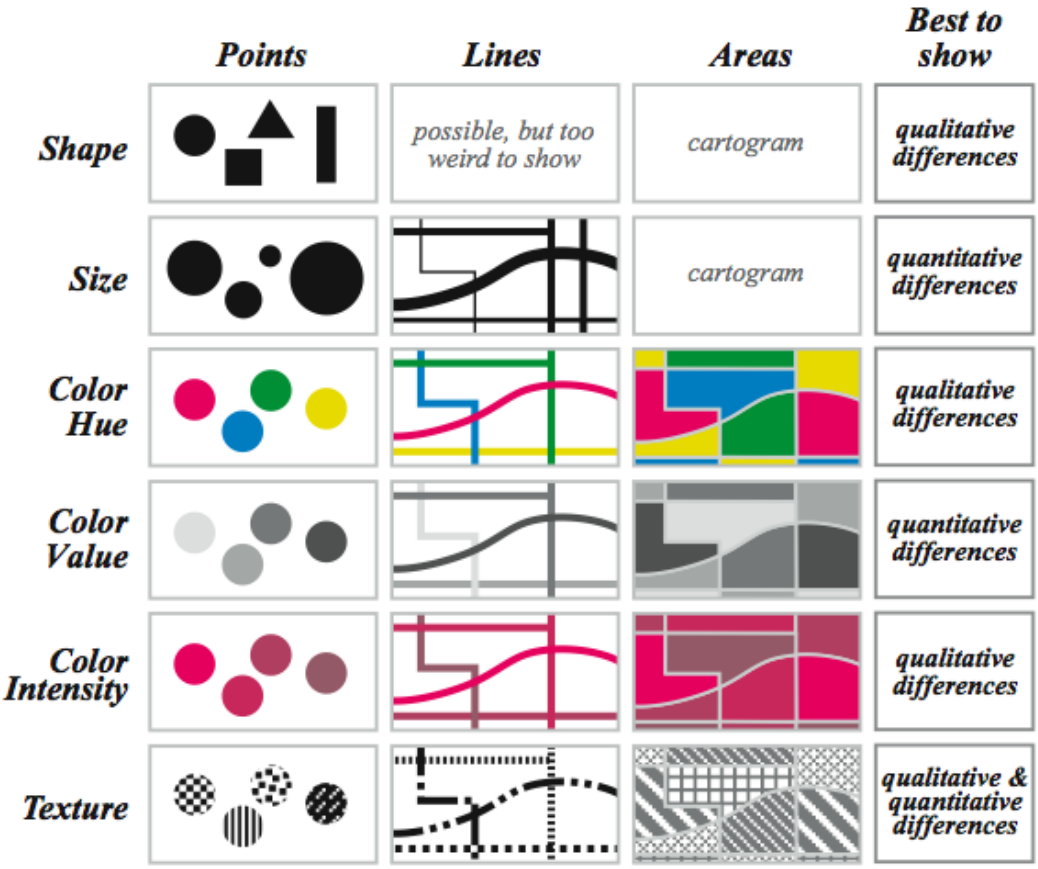


Figure 3: Retinal variables are a codification of how position, size, shape, color and texture are used to illustrate variations in the *components* of a visualization. This tabular form of Bertin's retinal variables is from Understanding Graphics [19] who reproduced it from *Making Maps: A Visual Guide to Map Design for GIS* [16]

Figure 3 illustrates common guidelines for encoding *components*, derived from what Bertin terms a retinal variable and most other visualization theorists call visual vari-

ables [**bertin˙semiology˙2011**, **krygier˙making˙2005**, **chambers˙graphical˙1983**, **wilkinson˙grammar˙2005**, **munzner˙visualization˙2014**]. The columns of figure 3 correspond to the type of observation: discrete points, continuous events (e.g. a timeseries), two dimensional continuous events (e.g. a vector map). The rows of figure 3 describe ways to visualize variations in the *components*; generally, quantitative components are represented by retinal variables that change quantitatively and categorical components are represented by retinal variables that vary qualitatively. In figure **??**, the hue of the marker is used to encode differentiation in species and the position of the marker is used to show variation in petal length and sepal length. The retinal variables suggest that any single visualization is limited to encoding at most about 8 or 9 components. Retinal variables provide guidelines for encoding *components*, but the choice of graph is based on the type and structure of the data.

2.1.2 Data Type and Structure

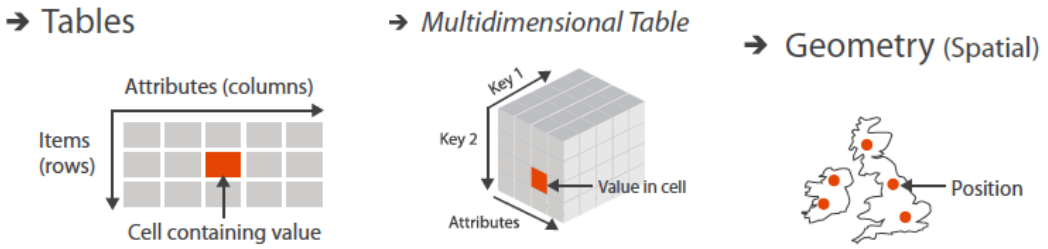


Figure 4: Keys are unique lookup values used to find individual observations in the dataset. Keys are positional references, and can be coordinates on a map or unique values such as a primary key in a database or a (time, latitude, longitude) index in a data cube. Image modified from a diagram from Munzner's *Visualization Analysis and Design* [24]

As shown in figure 2, there are multiple ways to translate data into pictures. A map is always an option, except when the observations do not have associated coordinates in a physical plane. Tamara Munzner provides a way to distinguish between these datasets using $\{key, value\}$ designations [24]. Munzner defines *values* as measurements of interest in the dataset, analogous to dependent variables in statistics. She defines *keys* as indexes that can be used to look up values, analogous to independent variables in statistics and dimensions

in computer science. Figure 4 illustrates how these keys are used to look up variables in a dataset:

- map: keys are the coordinates of the points
- table: row index, database primary key
- data cube: row, column, etc. (.e.g. i, j, k) index

Expanding on Munzner's key and value semantics, in many datasets the keys are discrete variables like time or geophysical locations sampled from a continuous curve, surface, or field. While these observations are discrete samples from the continuous space, often the continuous (functional) characteristic[22, 27] of the observational space is also of interest. Besides quantitative discrete, quantitative continuous, or categorical measurement type considerations, the choice of visualization is also influenced by the measurement being on an interval, ratio, nominal, or categorical scale.

Throughout this form we will be using the following conventions to discuss data:

variable

index

record

value

3 Topological Artist Model

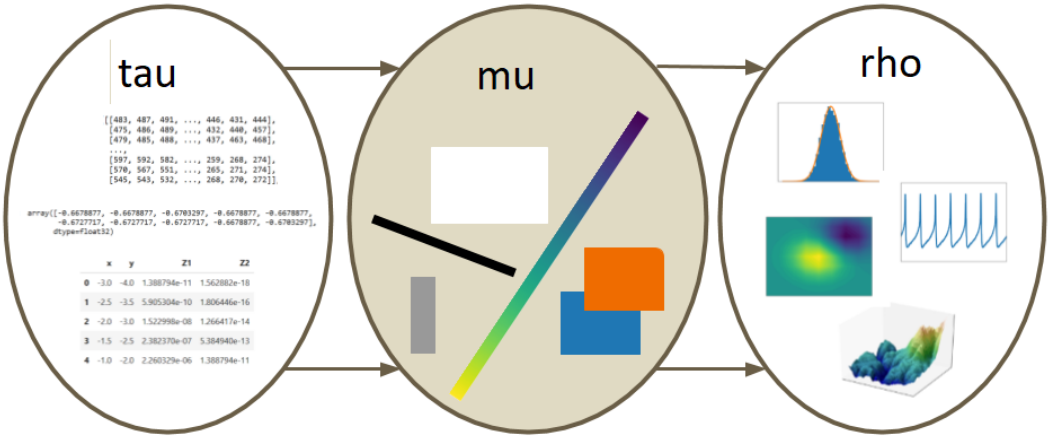


Figure 5: Visualization is equivariant maps between data and visual encoding of the variables and assembly of those encodings into a graphic. not gonna name these bubbles tau, mu, rho, but might keep the same basic structure of different types of data and encodings

should this be in 3rd person passive?

Visualization is generally thought of as structure preserving maps from data into graphics, and in this section we formally define that structure and how it is preserved via equivariant maps. We can then specify that a faithful visual mapping is structure preserving, and apply these constraints to visualizations we may want to develop or implement. We model the data, visual characteristic, and graphic stages of visualization, shown in figure 5, as topological structures that encapsulate types of variables and continuity; by doing so we can develop implementations that keep track of both in ways that let us distribute computation while still allowing assembly and dynamic update of the graphic.

We introduce a mathematical description of the visualization pipeline where artist $\mathscr{A}$ functions transform data space $\mathscr{E}$ to an intermediate representation in a prerendered graphic space $\mathscr{H}$.

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

We first describe how we model data(3.1), graphics(3.2), and intermediate visual characteristics (3.3) as fiber bundles. We then discuss the equivariant maps between data and visual characteristics (3.3.2) and visual characteristics and graphics (3.3.3) that make up the artist.

3.1 Data Space$E$

We build on Butler's proposal of using fiber bundles as a common data representation format for visualization data[5, 6] because fiber bundles are mathematical structures that are flexible enough express all the types of data described in section 2.1.2.

We model data as the fiber bundle (E, K, π, F), where E F and K are topological spaces that encode

F the properties of the variables in the fiber (3.1.1)

K the continuity of the records in the base space (3.1.3)

τ collections of records (3.1.4).

and E is the total space of data that F lives in. The bundle is the projection map π

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

that binds the variables F continuity K. The fiber bundles mentioned in this work are assumed to be trivial[18, 31], unless otherwise specified, because the trivial bundle is $E = K \times F$ such that extra structure in the total space E falls out and discussion can be focused on the fiber and base space.

3.1.1 Variables: Fiber Space F

The fiber is a topological space that is the set of possible values of the data; the values themselves can be any dimension and type and have any continuity. We use Spivak's description of simplicial database schemas [32] as the basis of our fiber space because he binds the components of the fiber to variable names and types. Spivak constructs a set $\mathbb{U}$ that

is the disjoint union of all possible objects of types $\{T_0, \ldots, T_n\} \in \mathbf{DT}$, where $\mathbf{DT}$ are the data types of the variables in the dataset. He then defines the single variable set $\mathbb{U}_\sigma$

$$\begin{array}{ccc} \mathbb{U}_\sigma & \longrightarrow & \mathbb{U} \\ {\scriptstyle \pi_\sigma}\downarrow & & \downarrow{\scriptstyle \pi} \\ C & \xrightarrow[\sigma]{} & \mathbf{DT} \end{array} \tag{3}$$

which is $\mathbb{U}$ restricted to objects of type T bound to variable name c. Given σ, the fiber for a one variable dataset is

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

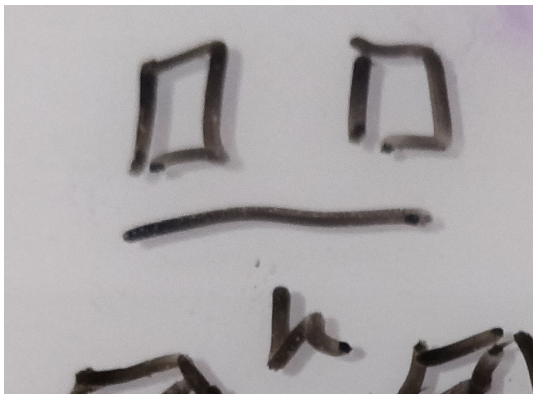
where σ is the schema binding variable name c to its datatype T. A dataset with multiple variables has a fiber that is the cartesian cross product of $\mathbb{U}_\sigma$ applied to all the columns:

$$F = \mathbb{U}_{T_1} \times \ldots \mathbb{U}_{T_i} \ldots \times \mathbb{U}_{T_n} \tag{5}$$

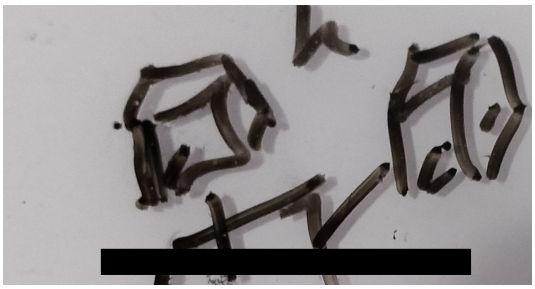
which is equivalent to

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

which allows us to decouple F into components F_i.



(a)



(b)

Figure 6: These two datasets have the same base space K but figure 6a has fiber $F = \mathbb{R} \times \mathbb{R}$ which is (time, temperature) while figure 6b has fiber $\mathbb{R}^+ \times \mathbb{R}^2$ which is (time, wind=(speed, direction))

For example, the data in figure 6a is a pair of times and °C temperature measurements taken at those times. Time is a positive number of type `datetime` which can be resolved to positive floats $\mathbb{U}_{\texttt{datetime}} = \mathbb{R}^+$. Temperature values are real numbers $\mathbb{U}_{\texttt{float}} = \mathbb{R}$. The fiber is

$$\mathbb{U} = \mathbb{R}^+ \times \mathbb{R} \tag{7}$$

where the first component F_0 is the set of values specified by ($c_0 = time, T_0 =$ `datetime`, $\mathbb{U}_\sigma = \mathbb{R}^+$) and F_1 is specified by ($c_1 = temperature$, $T_1 =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$). In figure 6b, temperature is replaced with wind. This wind variable is of type `wind` and has two components speed and direction $\{(s, d) \in \mathbb{R}^2 \mid 0 \leq s, 0 \leq d \leq 360\}$. Therefore, the fiber is

$$F = \mathbb{R}^+ \times \mathbb{R}^2 \tag{8}$$

such that F_1 is specified by ($c_1 = wind$, $T_1 =$ `wind`, $\mathbb{U}_\sigma = \mathbb{R}^2$)

3.1.2 Measurement Scales: Monoid Actions

After specifying F we next describe the ways in which we can transform the values by identifying the monoid actions M on the F. We use monoids as the abstraction because they encode composibilty, which maps well to the data transformation process in a software library [39].

A monoid [21] M_i is a set with an associative binary operator $* : M_i \times M_i \to M_i$. A monoid has an identity element $e \in M_i$ such that $e * a = a * e = a$ for all $a \in M_i$. A left monoid action [1, 30] of M_i is a set F_i with an action $\bullet : M \times F_i \to F_i$ with the properties:

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

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

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

$$M = M_0 \times \ldots \times M_i \times \ldots \times \ldots M_n \tag{9}$$

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

Steven's described the measurement scales[17, 34] in terms of the monoid actions on the measurements: nominal data is permutable, ordinal data is monotonic, interval data is translatable, and ratio data is scalable [37]. For example, given the fiber ($c = \textit{temperature}$, $T =$ `float`, $\mathbb{U}_\sigma = \mathbb{R}$) which is interval data:

- monoid operator addition $* = +$
- monoid operations: $f : x \mapsto x + 1$, $g : x \mapsto x + 2$
- monoid action operator composition $\bullet = \circ$

then the translation monoid actions on temperature satisfy the condition

$$\begin{array}{ccc} \mathbb{R} & & \\ {\scriptstyle x+1^\circ}\downarrow & \searrow^{(x+1^\circ)\circ(x+2^\circ)} & \\ \mathbb{R} & \xrightarrow[x+2^\circ]{} & \mathbb{R} \end{array} \tag{10}$$

where 1° and 2° are valid distances between two temperatures x.

3.1.3 Continuity: Base Space K

The advantage of fiber bundles is they provide a way to encode the continuity in a dataset as the base space K without making assumptions as to what that continuity is. In turn this representation of continuity can then be used to keep track of how the data fits together, for example if a visualization of a very large dataset calls for parallelization.

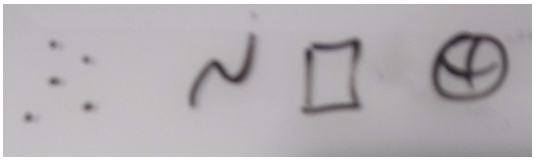


Figure 7: The topological base space K encodes the connectivity of the data space, for example if the data is independent points or a map or on a sphere

As illustrated in figure 7, K is akin to an indexing space into E that describes the structure of E. K can have any number of dimensions and can be continuous or discrete.

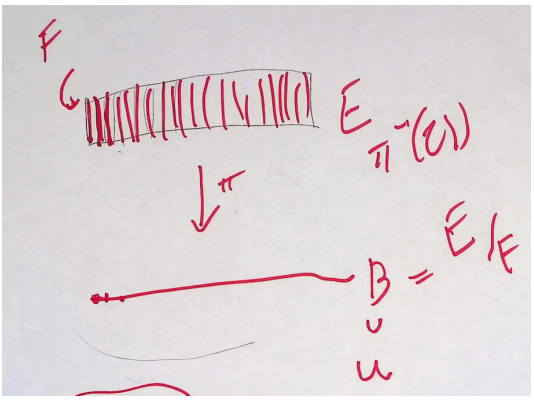


Figure 8: The base space E is divided into fiber segments F. The base space K acts as an index into the records in the fibers. this figure might be good all the way up top to lay out the components of fb

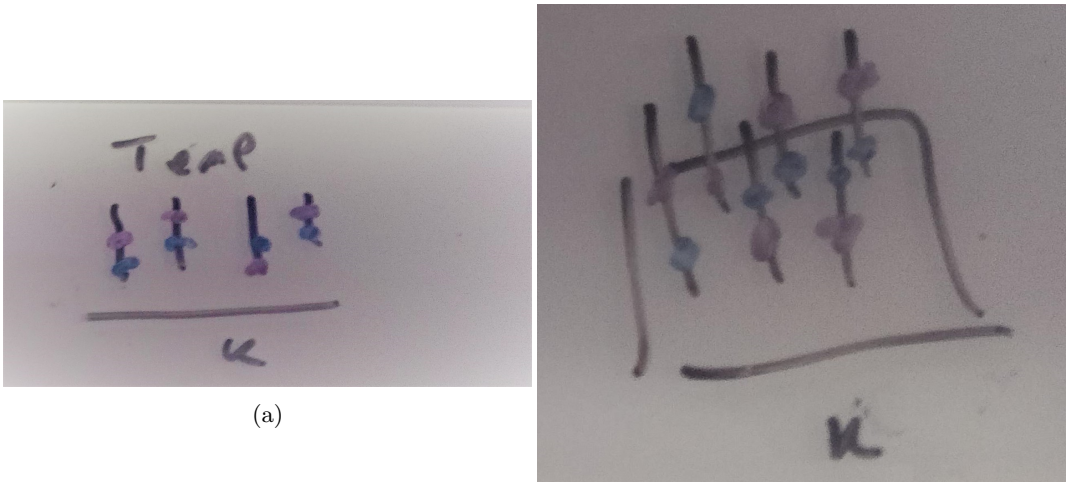
Formally K is the quotient space [26] of E meaning it is the finest space[2] such that every $k \in K$ has a corresponding fiber F_k[26]. In figure 8, E is a rectangle divided by vertical fibers F, so the minimal K for which there is always a mapping $\pi : E \to K$ is the line.

As with fibers and monoids, we can decompose the total space into components $\pi : E_i \to K$ where

$$\pi : E_1 \oplus \ldots \oplus E_i \oplus \ldots \oplus E_n \to K \tag{11}$$

which is a decomposition of F. The K remains the same because the connectivity of records does not change just because there are fewer elements in each record.

The datasets in figure 9 have the same fiber of (temperature, time). In figure 9a the fibers lie over discrete K such that the records in the datasets in the fiber bundles are discrete. The same fiber in figure 9b lies over a continuous interval Ksuch that the records are samples from a continuous function defined on K.



(a)

(b)

Figure 9: These two datasets have the same (time, temperature) fiber. In figure **??** the total space E is discrete over points $k \in K$, meaning the records in the fiber are also discrete. In figure **??** E lies over the continuous interval K, meaning the records in the fiber are sampled from a continuous space. revamp figure: F=Plane, k1 = dots, k2=line

201 ### 3.1.4 Data: Sections τ

While the fiber and base space describe the general structure of all data that lives in the fiber bundle, the sections $\tau : K \to E$ define the datasets that live in the fiber. We generalize Spivak's description of the section as a table of records [32] to any sort of structured dataset such that

$$\begin{array}{ccc} F & \hookrightarrow & E \\ & & \pi \downarrow \uparrow \tau \\ & & K \end{array} \tag{12}$$

such that there is always a map $\pi(\tau(k)) = k$. There can be many sections τ; the space of global sections is $\Gamma(E)$. For a trivial fiber bundle, the section is

$$\tau(k) = (k, (g_{F_0}(k), \ldots, g_{F_n}(k))) \tag{13}$$

where $g : K \to F$ is the index function into the fiber. Because we can decompose the bundle and the fiber, we can formulate τ as

$$\tau = (\tau_0, \ldots, \tau_i, \ldots, \tau_n) \tag{14}$$

where each section τ_i is a variable or set of variables.

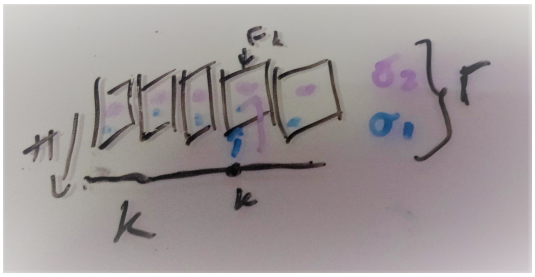


Figure 10: Fiber (time, temperature) with an interval K basespace. The sections τ_i and τ_j are constrained such that the time variable must be monotonic, which means each section is a timeseries of temperature values. They are included in the global set of sections $\tau_1, \tau_2 \in \Gamma(E)$

In the example in figure 10, the fiber is (*time*, *temperature*) as described in figure 6 and the base space is the interval K. The section τ_i resolves to a series of monotonically increasing in time records of (time, temperature) values. Section τ_j returns a different timeseries of (time, temperature) values. Both sections are included in the global set of sections $\tau_1, \tau_2 \in \Gamma(E)$.

3.1.5 Sheaf and Stalk

Often a graphic may need to be updated with live data or support zooming in on a segement of the dataset; to support working with a subset of data, we can use the sheaf $\mathcal{O}(E)$. All fiber bundles are locally trivial, which means that E restricted over a small enough neighborhood $U \subset K$ is a locally trivial bundle over U[18]. The sheaf $O(E)$ is the localized section of fibers $\iota^*\tau : U \to \iota^* E$

$$\begin{array}{ccc} \iota^* E & \xleftarrow{\iota^*} & E \\ {\scriptstyle\pi}\downarrow\uparrow{\scriptstyle\iota^*\tau} & & {\scriptstyle\pi}\downarrow\uparrow{\scriptstyle\tau} \\ U & \xleftarrow{\iota} & K \end{array} \tag{15}$$

pulled back over the neighborhood U via the inclusion map $\iota : U \to K$. The localized section is the germ $\xi^*\tau$. The neighborhood of points k_i surrounding the point k the sheaf lies over is the stalk $\mathcal{F}_b$ [31, 33]. While E is only the fiber F_k over a specific k, the stalk includes nearby records because the sheaf lies over the neighborhood U. While this can be useful for visual transforms, often the extra needed information can be found in the smaller jet bundle $\mathcal{J}$ [15, 25]. For example, line thickness requires the derivative of the given position to be rendered, which can be found in $E' = E + \mathcal{J}(E)$

3.2 Graphic: H

We can separate the structure of the graphic from the properties of the output format by modeling the space of graphics as a fiber bundle (H, S, π, D). As with data, the fiber bundle is for a class of graphics with shared base space S(3.2.1) and fiber D(3.2.2) and the sections ρ(3.2.3) encode a graphic where the visual characteristics are fully specified.

3.2.1 Idealized Display D

The fiber D is an idealized infinite resolution version of the target display space, for example a 2D screen or 3D printer. In this work, we assume a 2D opaque image $F = \mathbb{R}^5$ with elements

$$(x,\ y,\ r,\ g,\ b) \in D \tag{16}$$

such that a rendered graphic only consists of 2D position and color. To support overplotting and transparency, the fiber could be $F = \mathbb{R}^7$ such that $(x, y, z, r, g, b, a) \in D$ specifies the target display. The location coordinates x and y are defined in terms of the display, while the (r, g, b) values are filled in via a lookup on S.

3.2.2 Continuity of the Graphic S

An assumption of graphical representations is that they match the continuity of the data[10, 35], but the underlying topology S of a graphic may need more dimensions than the data topology K so that the glyph can be defined in F. Therefore we define the base space

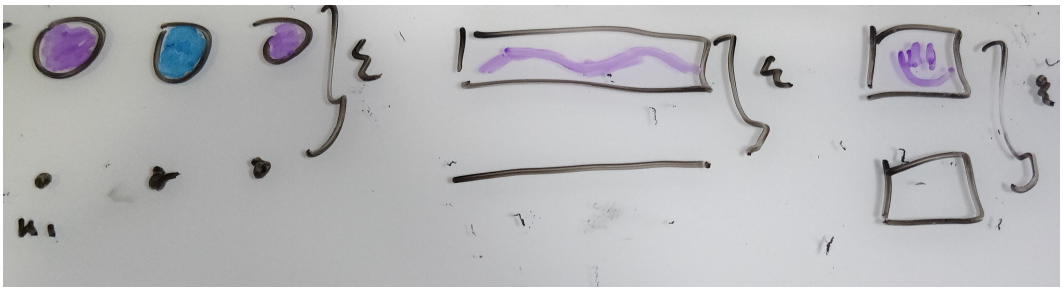


Figure 11: The scatter and line graphic base spaces have one more dimension of continuity than K so that S can encode physical aspects of the glyph, such as shape (a circle) or thickness. The heatmap has the same continuity in the graphic S as in the data K. add α, β coordinates to figures

mapping from graphic $S$to data K

$$\begin{array}{ccc} E & & H \\ {\scriptstyle\pi}\downarrow & & {\scriptstyle\pi}\downarrow \\ K & \xleftarrow{\xi} & S \end{array} \tag{17}$$

as the deformation retraction [28] $\xi : S \to K$ that goes from a region $s \in S_k$ to its associated point s, such that when $\xi(s) = k$, $\xi^*\tau(s) = \tau(s)$. While dimensions can be added to S, it retains the same continuity as K.

In figure 11 each disk S_k indexes how elements in D are glued together to generate a single glyph that is the visual representation of a single record in F_k. For the line, the region β over a point α_i specifies the thickness of the line in S for the corresponding τ on K. The heatmap has the same continuity in data space and graphic space such that no extra dimensions are needed.

3.2.3 Renderable Glyph ρ

A section $\rho : S \to H$ defines a piece of the graphical representation of the data. Evaluated on a single s ρ returns a single element in $D\dot{\text{F}}$or a 2D screen, the pixel is defined as a region $p = [y_{top}, y_{bottom}, x_{right}, x_{left}]$ of the rendered graphic. Since the x and y in p are in the same coordinate system as the x and y components of D the inverse map of the bounding

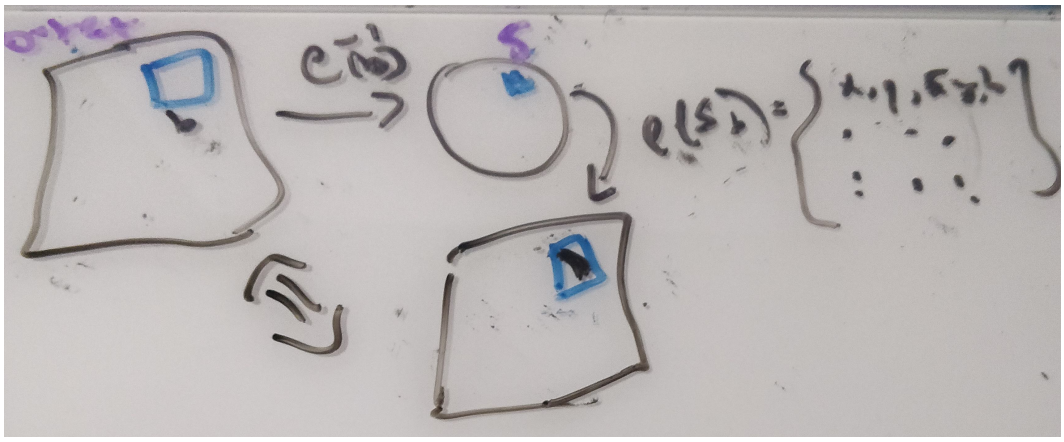


Figure 12: To render a graphic, a pixel p is selected in the display space, which is defined in the same coordinates as the x and y components in DThe inverse mapping ${\rho_{xy}}^{(}p)$ returns a region $S_p \subset S$. $\rho(S_p)$ returns the list of elements $(x,\ y,\ r,\ g,\ b) \in D$ that lie over S_p. The integral over the $(r,\ g,\ b)$ elements is the color of the pixel.

box $S_p = {\rho_{xy}}^{-1}(p)$ is a region $S_p \subset S$. Integrating over this region on S

$$r_p = \iint_{S_p} \rho_r(s) ds^2 \tag{18}$$

$$g_p = \iint_{S_p} \rho_g(s) ds^2 \tag{19}$$

$$b_p = \iint_{S_p} \rho_b(s) ds^2 \tag{20}$$

yields the color of the pixel p.

As shown in figure 12, the output space queries into the graphic bundle to render the image. We select a pixel p in the output space, inverse map the region of the pixel into $S_p \subset S$, then compute the section $gsection$ over the region S_p. The section yields the set of elements in D that specify the (r, g, b) values corresponding to the region p. The color of the pixel is then obtained by taking the integral of $\rho_{rgb}(S_p)$.

3.3 Artist

The artist is the function that converts data into graphics; its name is taken from the analogous part of Matplotlib[14] that builds visual elements to pass off to the renderer. The artist A is a mapping from E padded with data from $\mathcal{J}(E)$ to a graphic that is a section ρ in $\Gamma(H)$

$$\begin{array}{ccccccc} E' & \xrightarrow{\nu} & V & \xleftarrow{\xi^*} & \xi^*V & \xrightarrow{Q} & \Gamma(H) \\ & \searrow^{\pi} & \downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \xi^*\pi} & \swarrow_{\pi} & \\ & & K & \xleftarrow{\xi} & S & & \end{array} \tag{21}$$

with an intermediate fiber bundle V to hold visual representations and stages

1. ξ binding the continuity in the graphic to the continuity in the data (3.2.2)
2. ν conversion of data into visual characteristics (3.3.2)
3. Q assembly of visual variables into a glyph (3.3.3)

of the visual transformation illustrated in figure 5. The functions ξ ν and Q are defined such that they can be evaluated on a single section τ, which allows the artist to be implemented such that it does not need all the data. This allows for artists tuned to distributed and streaming data.

3.3.1 Visual Fiber Bundle V

The visual fiber bundle $(V,\ K,\ \pi,\ P)$ has section $\mu : V \to K$ that resolves to a visual variable [3, 23] in fiber P. The fiber space P is defined in terms of the parameters of the visualization specification- for example aesthetics in ggplot [38], channels in vega[29] or parameters in VTK[11] and Matplotlib.

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

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

Table 1 is a sample of the fiber space for Matplotlib [13]. A section $\mu$is a tuple of visual values that specifies the visual characteristics of a part of the graphic. For example, given a fiber of $\{xpos, ypos, color\}$ one possible section could be $\{.5, .5, (255, 20, 147)\}$. The $codomain(\nu_i)$ determines the monoid actions on μ_i. These fiber components are implicit in the library, by making them explicit as components of the fiber we can build consistent definitions and expectations of how these parameters behave.

3.3.2 Visual Channels

As introduced in section 2.1.1, there are many ways to encode data visually. We define the visual transformers ν as the set of independent conversion functions

$$\{\nu_0, \ldots, \nu_n\} : \{\tau_0, \ldots, \tau_n\} \mapsto \{\mu_0, \ldots, \mu_n\} \tag{22}$$

where $\nu_i : \tau_i \mapsto \mu_i$ is an equivariant map such that there is a monoid homomorphism from F_i to $vfiber_i$. A validly constructed ν is one where the diagram of the monoid transform m

$$\begin{array}{ccc} E_i & \xrightarrow{\nu_i} & V_i \\ {\scriptstyle m_x}\downarrow & & \downarrow{\scriptstyle m_v} \\ E_i & \xrightarrow{\nu_i} & V_i \end{array} \tag{23}$$

commutes such that $\nu_i(m_x(E_i)) = m_v(\nu_i(E_i))$. This equivariance constraint yields guidance on what makes for an invalid transform. For example, the conversion $\nu_i(x) = .5$ does not commute under translation monoid action $t(x) = x + 2$

$$\nu(t(x+2)) \stackrel{?}{=} \nu(x) + \nu(2) \tag{24}$$

$$.5 \neq .5 + .5 \tag{25}$$

On the other hand figure **??** illustrates a valid ν mapping from **Strings** to symbols. The group action on these sets is permutation, so shuffling the words must have an equivalent shuffle of the symbols they are mapped to. To preserve ordinal and partial order monoid actions, ν must be a monotonic function such that given $x_1, x_2 \in E_i$, if $delement_1 \leq delement_2$ then $\nu(x_1) \leq \nu(x_2)$. For interval scale data, ν is equivariant under translation monoid actions if $\nu(x + c) = \nu(x) + \nu(c)$. For ratio data, there must be equivalent scaling $\nu(xc) = \nu(x)\nu(c)$.

3.3.3 Assembling Marks

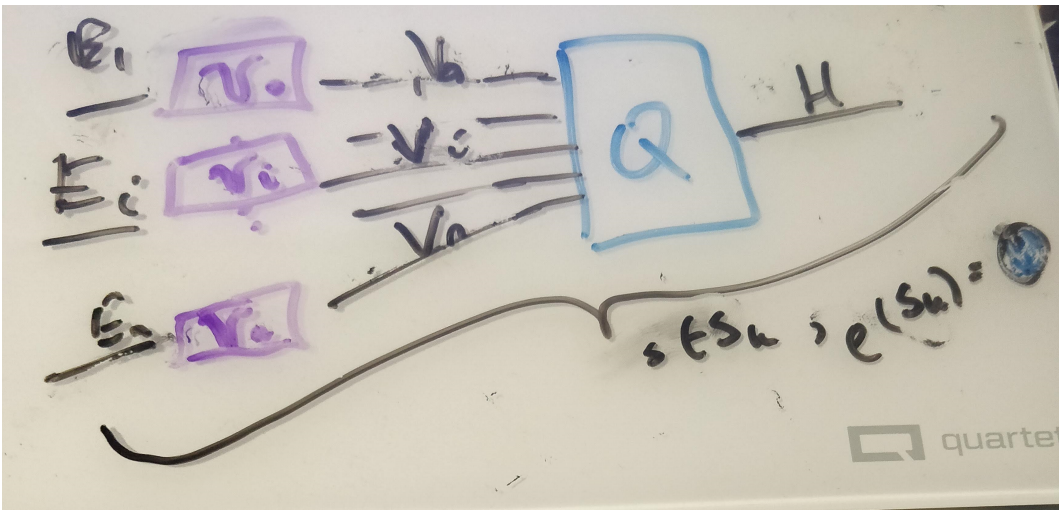


Figure 13: ν functions convert data τ_i to visual characteristics μ_i, then Q assembles μ_i into a graphic ρ such that there is a map ξ preserving the continuity of the data. ρ applied to a region of connected components S_j generates a graphical mark.

As shown in figure 13, the assembly function Q combines the fiber F_i wise ν transforms into a single glyph. Together, ν and Q are a map-reduce operation: map the data into their visual encodings, reduce the encodings into a glyph. As with ν the constraint on Q is that for every monoid actions on the input μ there is a corresponding monoid action on the output ρ.

Since we define the equivariant map as $Q : \mu \mapsto \rho$, we define an action on the subset of graphics $Q(\Gamma(V)) \in \Gamma(H)$ that Q can generate. We then define the constraint on Qsuch that if Q is applied to μ, μ' that generate the same ρ then the output of both sections acted on by the same monoid m must be the same.

Lets call the visual encodings $\Gamma(V) = X$ and the graphic $Q(\Gamma(V)) = Y$. If for all monoids $m \in M$ and for all $\mu, \mu' \in X$, the output is equivalent

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \tag{26}$$

then a group action on Y can be defined as $m \circ \rho = \rho'$. The transformed graphic ρ' is equivariant to a transform on the visual bundle $\rho' = Q(m \circ \mu)$ on a section that $\mu \in Q^{-1}(\rho)$ that must be part of generating ρ.

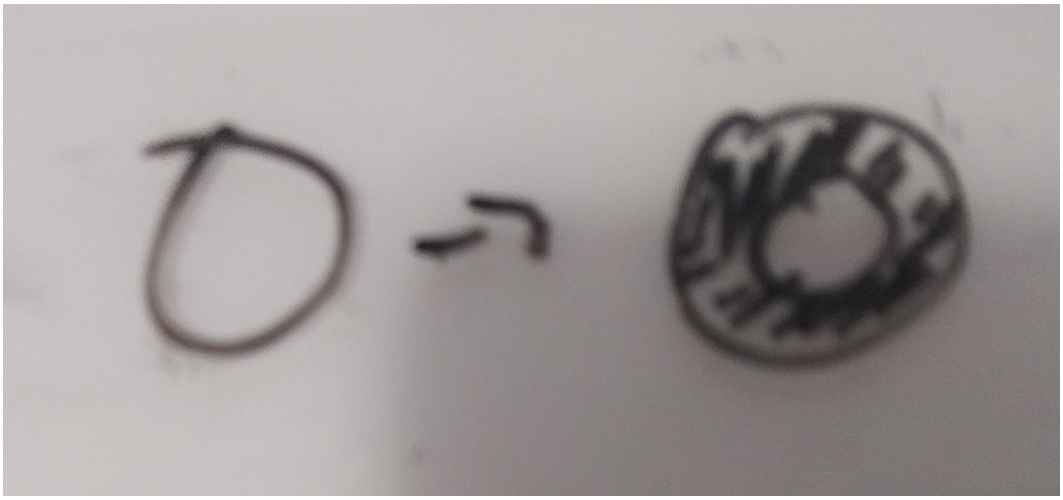


Figure 14: These two glyphs are generated by the same Q function, but differ in the value of the edge thickness parameter μ_i. A valid Q is one where a shift in μ_i is reflected in the glyph generated by ρ.

The glyph in figure 14 has the following characterstics Pspecified by ($xpos$, $ypos$, $color$, $thickness$) such that one section is $\mu = (0, 0, 0, 1)$ and $Q(\mu) = \rho$ generates a piece of the thin hollow circle. The equivariance constraint on $Q$is that the action $m = (e, e, e, x + 2)$, where e is identity, applied to μ such that $\mu' = (e, e, e, 3)$ has an equivalent action on ρthat causes $Q(\mu')$ to be equivalent to the thicker circle in figure 14.

DEGENERATE Q - drawing a blank if this is necessry and if so how

Check a well defined map $M \times Y \to Y$

To output a mark [3, 7], Q is called with all the regions s that map back to a set of connected components $J \subset K$:

$$J = \{j \in K \text{ s. t. } \exists \gamma \text{ s.t. } \gamma(0) = k \text{ and } \gamma(1) = j\} \tag{27}$$

where the path[8] γ from k to j is a continuous function from the interval [0,1]. We define the mark as the graphic generated by $Q(S_j)$

$$H \underset{\rho(S_j)}{\overset{}{\rightleftarrows}} S_j \underset{\xi^{-1}(J)}{\overset{\xi(s)}{\rightleftarrows}} J_k \tag{28}$$

such that for every mark there is at least one corresponding section on K.

3.3.4 Sample Qs

In this section we formulate the minimal Q that will generate distinguishable graphical marks: non-overlapping scatter points, a non-infinitely thin line, and a heatmap.

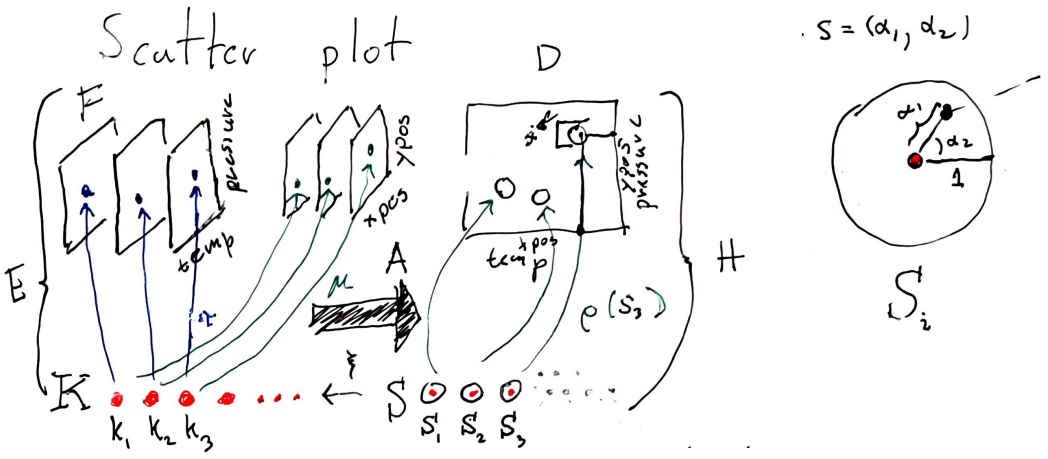


Figure 15: The data is discrete points (temperature, time). Via ν these are converted to (xpos, ypos) and pulled over discrete S. These values are then used to parameterize ρwhich returns a color based on the parameters (xpos,ypos) and position α, β on S_k that ρ is evaluated on.

The scatter plot in figure **??** can be defined as $Q(xpos, ypos)(\alpha, \beta)$ where color $\rho_{RGB} = (0, 0, 0)$ is defined as part of Q and $s = (\alpha, \beta)$ defines the region on S. The position of this swatch of color can be computed relative to the location on the disc S_k as shown in figure 15:

$$x = size \bullet \alpha \bullet \cos(\beta) + xpos \tag{29}$$

$$y = size \bullet \alpha \bullet \sin(\beta) + ypos \tag{30}$$

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

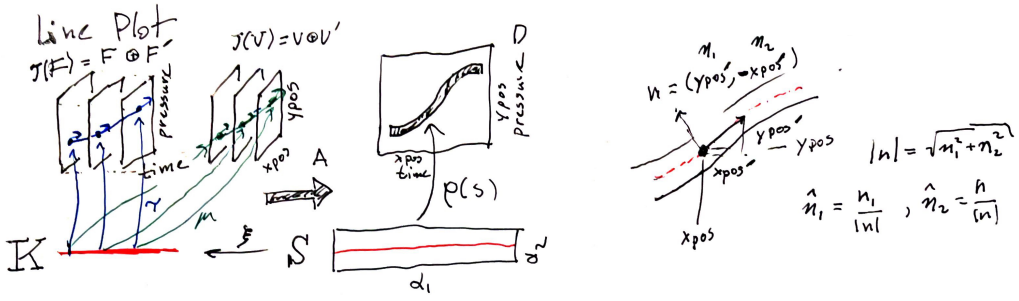


Figure 16: The line fiber (*time*, *temp*) is thickend with the derivative (*time*′, *temperature*′ because that information will be necessary to figure out the tangent to the point to draw a thick line. This is because the line needs to be pushed perpendicular to the tangent of (xpos, ypos). this is gonna move once this gets regenerated w/ labels The data is converted to visual characteristics (xpos, ypos). The α coordinates on S specifies the position of the line, the β coordinate specifies thickness.

The line plot $Q(xpos, \hat{n_1}, ypos, \hat{n_2})(\alpha, \beta)$ shown in fig 15 exemplifies the need for the jet discussed in section **??**. The line needs to know the tangent of the data to draw an envelope above and below each (xpos,ypos) such that the line appears to have a thickness. The magnitude of the thickness is

$$|n| = \sqrt{{n_1}^2 + {n_2}^2} \tag{31}$$

such that the normal is

$$\hat{n_1} = \frac{n_1}{|n|}, \ \hat{n_2} = \frac{n_2}{|n|} \tag{32}$$

which yields components of ρ

$$x = xpos(\xi(\alpha)) + \beta\hat{(}n_1)(\xi(\alpha)) \tag{33}$$

$$y = ypos(\xi(\alpha)) + \beta\hat{(}n_2)(\xi(\alpha)) \tag{34}$$

where (x,y) look up the position $\xi(\alpha)$ on the data and then apply thickness β at that location.

Q: heatmap

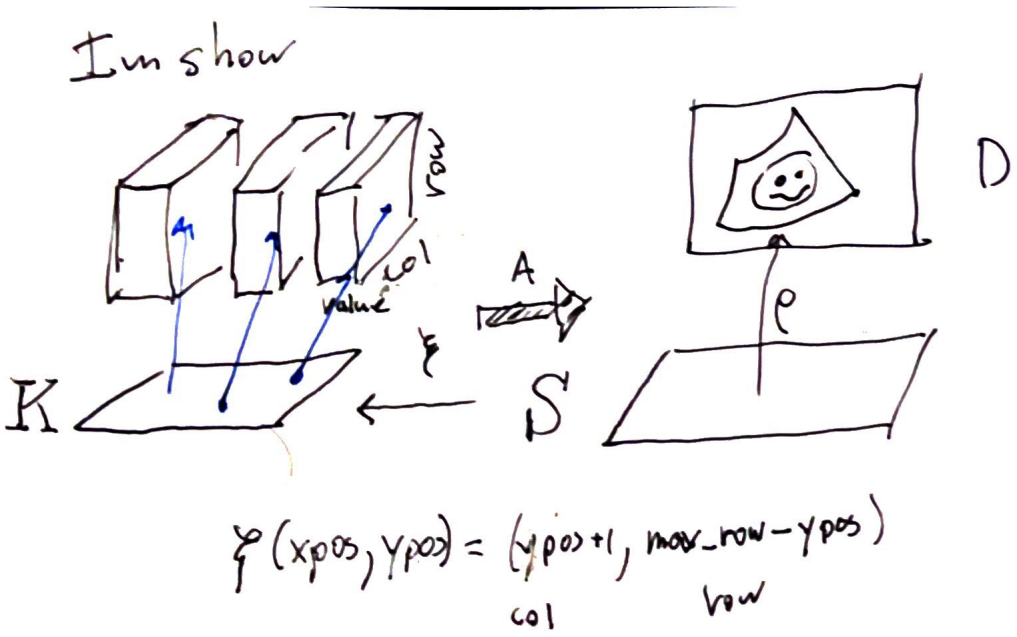


Figure 17: The only visual parameter a heatmap requires is color since ξ encodes the mapping between position in data and position in graphic.

The heatmap $Q(color)$ in figure 17 is a direct lookup $\xi : S \to K$ such that

$$R = R(\xi(\alpha, \beta)) \tag{35}$$

$$G = G(\xi(\alpha, \beta)) \tag{36}$$

$$B = B(\xi(\alpha, \beta)) \tag{37}$$

where ξ may do some translating to a convention expected by Q for example reorientng the array such that the first row in the data is at the bottom of the graphic.

Figure 18: Each of these graphics is generated by a different artist Awhich is the equivalance class of scatter plots A'this is gonna be a whole bunch of scatter plots

3.3.5 Equivalance class of artists A'

As formulated above, every artist function A has fixed ν and Q which generates a distinct graphic ρ. It is impractical to implement an artist for every single graphic; instead we implement the equivalence class of artists $\{A \in A' : A_1 \equiv A_2\}$. Equivalent artists have the same fiber bundle V and same assembly function Q but act on different sections μ. To further simplify implementation, we identify a minimal P associated with each A' that defines what visual characteristics of the graphic must originate in the dataneeds citation, maybe friendlys history or acquired codes of meaning. For example, a scatter plot of red circles is the output of one artist, a scatter plot of green squares the output of another, as are the rest of the graphs in figure **??**. These two artists are equivalent since their only difference is in the literal visual encodings (color, shape). Shape and color could also be defined in Q but the position must come from the fiber $P = (xpos, ypos)$ since fundementally a scatter plot is the plotting of one position against another[10]. We also use this criteria to identify derivative types, for example the bubble chart[35] is a type of scatter where by definition the glyph size is mapped from the data.

3.4 Making the fiber bundle computable

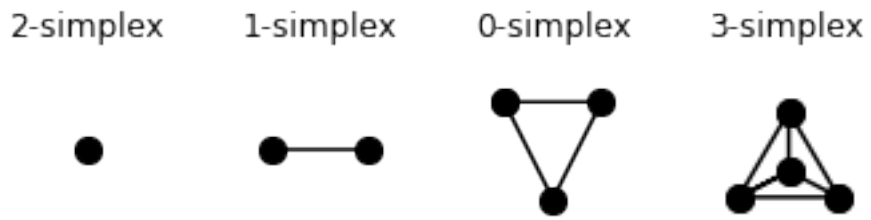


Figure 19: Simplices can encode the connectivity of the data, from fully disconnected (0 simplex) records to all records are connected to at least 3 others

One way of expressing the connectivity of records in a dataset is to implement K as a simplacial complex, which is a set of simplicies such as those shown in figure 19. The advantage of triangulization is that it is general enough to work for more complex topology based visualization methods [12] while also providing a consistent interface of vertices, edges, and faces for ξ to map into. When triangulated, the simplices encode the continuity in the data

simplex	**continuity**	τ
vertex	discrete	$\tau(k)$
edge	1D	$\tau(k, \alpha)$
face	2D	$\tau(k, \alpha, \beta)$

Table 2

such that each section is bound to a simplex $k \in K$. As shown in table 2, in a 1D continuous spaces each τ lies distance α along edge k, while in a 2D continuous space each τ lies at coordinate α, β on the face k. This is directly analogous to indexing to express connectivity in N-D arrays, while also natively supporting graphs and trees as they are simplacial complices of nodes and edges. Path connected components are then sections where edges or faces meet.

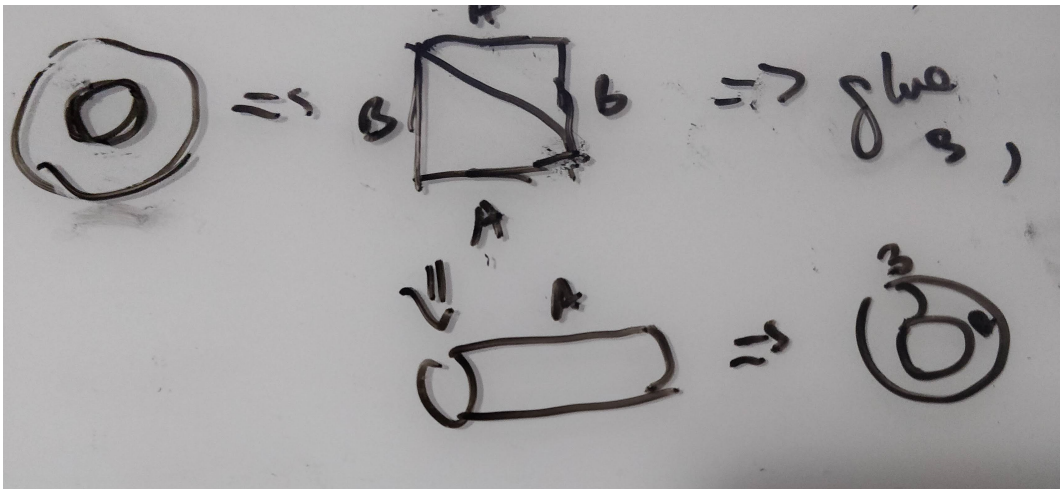


Figure 20: The torus $E$is unraveled into a simplacial complex of 2 faces K. Transition functions are defined on the edges of Ksuch that surface can be glued back into the torus. add cross sections a and b to ring and color same as edges in complex

One way of encoding the torus in figure 20 while retaining the continuity of both cross sections a, b is to unravel it into a simplacial complex of two triangles with labeled edges. Transition functions δ are defined on the edges such that a can be glued to a' and b to b' to reconstruct the torus. This simplacial complex is then used as the base space encoding the continuity of data that lies in the torus. A constraint on the transition functions is that the monoid actions on the fibers on the edges of E are commutative $M * F \mapsto \delta(MF) = M * \delta(F)$

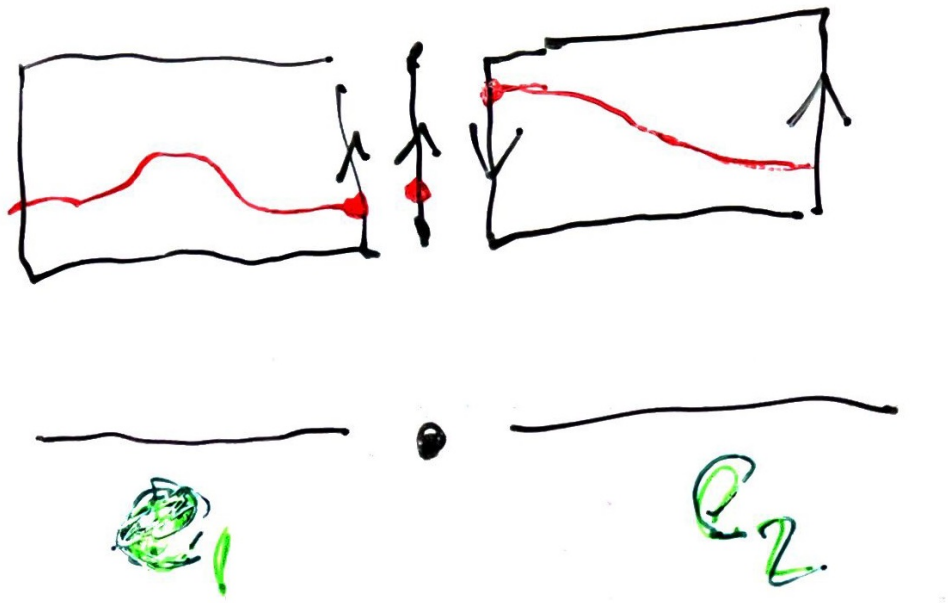


Figure 21: Many non-trivial spaces can be made locally trivial by dividing Einto locally trivial subspaces and defining transition functions between the edges on $K$for how to glue the two subspaces such that the $\tau$are continuous.

Another advantages of triangulization is that it provides a way to encode non-trivial structures such as the mobius strip[20]. As shown in figure 21, one way of making the mobius strip trivial is to seperate it into two spaces E_1 and E_2 and then define transition functions that specify that the edges of E_1 need to be reversed to line up with E_2 such that the sections along the edges meet. As with the torus, the transition functions must preserve monoid commutativity.

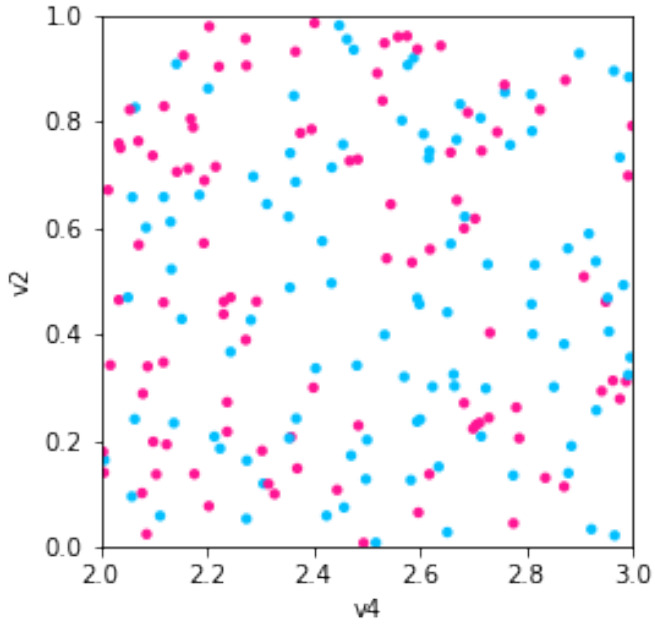
4 Matplottoy

We build on the existing Matplotlib architecture [13] so that we can initially focus on the data to graphic transformations and rely on Matplotlib for the other graphical elements of the visualization and the rendering. We first introduce an implementation of scatter, line, and bar charts because they map to the fundemental marks of point, line, and area. We then introduce aggregated bar charts to show how to build more complex graphics.

Because it is impractical for the user to specify every possible visual parameter, we define a required set of visual variables that must be explicitly specified to generate distinguishable marks.

4.1 Scatter

A scatter plot [9, 36] is a chart type for plotting discrete values against each other. Minimally a scatter plot requires and x or y position, but the other variables in our dataset can be mapped to visual aspects of the scatter graphical mark.



For the scatter plot in figure **??**, we define Q as

$$Q(\mu_{xpos}, \mu_{ypos}, \mu_{facecolor}, \mu_s) \tag{38}$$

which we implement building on the `matplotlib.collections` API:

```
    class Point(mcollections.Collection):
        # this is the visual fiber
        required = {'x', 'y'}
        optional = {'facecolors', 's'}
        def __init__(self, data, transforms, *args, **kwargs):
            # check that the constraints on Q can be met
        def draw(self, renderer, *args, **kwargs):
            # assemble the glyph
            super().draw(renderer, *args, **kwargs)
```

(subject to change). Unlike equation 38, we pass also pass a reference to the data so that getting the data can be fully curried until absolutely necessary. The `*args` and `**kwargs` are artifacts of using Matplotlib as a base. We pass in the μ as the dictionary "transforms" mostly for readability, which is also why we define the visual fiber as class attributes.

Inside the `__init__` constructor, is a check if there is a valid mapping between S and K. Line 7-8 check if the user has passed in the required μ and whether the ν functions are valid for the input data.

- assume that the ν are inherently valid - no error checking - don't try to exactly encode monoid - is it a key feature you want to encode - What is the important bits?

```
def __init__(self, data, transforms, *args, **kwargs):
    super().__init__(*args, **kwargs)
    # check that the data you're trying to transform
    # has a way to provide vertex data
    assert 'vertex' in data.FB.K['tables'] #there is no check here

    # check that you've given the required parameters
    utils.check_constraints(Point, transforms.keys())
    utils.validate_transforms(data.FB.F, transforms)

```

```
11      self.data = data
12      self.transforms = transforms
```

372 Lines 9-10 attach the data and transforms to the *Point* object as a concession to the
373 Matplotlib architecture that seperates drawing from creation. In `draw`, the attributes of the
374 graphic are set to the different sections of the visual fiber bundle.

```
1  def draw(self, renderer, *args, **kwargs):
2      # use \xi^{-1} because we want to pull forward the data
3      # propogate information of the downselection to do that
4      # restrict sheaf to subset of data over current view - feedback at draw restriction
5      view = self.data.view('vertex') #resolve to size \xi^{-1}
6
7      # \nu(\tau)
8      visual = utils.convert_transforms(view, self.transforms)
9
10     # assembles taus to generate idiom
11     visual['s'] = itertools.repeat(visual.get('s', 0.05))
12     visual['facecolors'] = visual.get('facecolors', "C0")
13     #switch out to a marker
14     self._paths = [mpath.Path.circle(center=(x,y), radius=s)
15                    for (x, y, s)
16                    in zip(visual['x'],visual['y'], visual['s'])]
17
18     self.set_facecolors(visual['facecolors'])
19     super().draw(renderer, *args, **kwargs)
```

375 Line 1 pulls back into the data bundle and returns $\tau = \{\tau_0, \ldots, tau_n\}$. Line 3 applies
376 the transforms ν to τ to build the visual bundle V. Line 8 sets the abstract segment marks
377 with the visual characteristics. Line 10 passes this information to the renderer, acting like
378 a *rho*.

The visual fiber bundle is specified as a dictionary {parameter : (variable, ν), parameter : value}

```
transforms = {'y': ('v2', position.Identity()),
              'x': ('v4', position.Identity()),
              'facecolors': ('v3', color.Categorical(
                                  {'true':'deeppink',
                                   'false':'deepskyblue'})),
              's':.01}
```

where the ν functions are represented as classes so that parameters can be curried. For example, `color.Categorical` is a whole set of ν functions explicitly tuned to specific categorical mappings. A pure functional approach would mean the user would have to write new functions

```
def color_true_deepink_false_deepink(value):
    return {'true':'deeppink', 'false':'deepskyblue'}[value]
```

for every categorical colormapping, which would be somewhat untenable. An alternative approach could be to partially curry:

```
     'facecolors': ('v3', (color.categorical, {'true':'deeppink', 'false':'deepskyblue'}))
```

but that doesn't allow for as easy reuse? (I should probably actually just switch to this form) The `color.categorical` function is implemented as

```
class Categorical:
    def __init__(self, mapping):
        """goal of init is to store parameters that would otherwise be
        curried higher level function"""
        assert(mcolors.is_color_like(color) for color in mapping.values())
        self._mapping = mapping

```

```
    def convert(self, value):
        values = np.atleast_1d(np.array(value, dtype=object))
        return [mcolors.to_rgba(self._mapping[v]) for v in values]

    def validate(self, mtype):
        return mtype in ['nominal']
```

where the `convert` function converts the input value to the internal normalized form Matplotlib expects. The `validate` function checks which monoid actions the transform supports; optimally this method should be replaced by functions that check the properties, for example a `is_ordinal` method that checks that monotonicity is preserved.

References

[1] *Action in nLab*. https://ncatlab.org/nlab/show/action#actions_of_a_monoid.

[2] Professor Denis Auroux. "Math 131: Introduction to Topology". en. In: (), p. 113.

[3] Jacques Bertin. "II. The Properties of the Graphic System". English. In: *Semiology of Graphics*. Redlands, Calif.: ESRI Press, 2011. ISBN: 978-1-58948-261-6 1-58948-261-1.

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

[5] D. M. Butler and M. H. Pendley. "A Visualization Model Based on the Mathematics of Fiber Bundles". en. In: *Computers in Physics* 3.5 (1989), p. 45. ISSN: 08941866. DOI: 10.1063/1.168345.

[6] David M. Butler and Steve Bryson. "Vector-Bundle Classes Form Powerful Tool for Scientific Visualization". en. In: *Computers in Physics* 6.6 (1992), p. 576. ISSN: 08941866. DOI: 10.1063/1.4823118.

[7] Sheelagh Carpendale. *Visual Representation from Semiology of Graphics by J. Bertin*. en.

[8] "Connected Space". en. In: *Wikipedia* (Dec. 2020).

[9] M. Friendly. “A Brief History of Data Visualization”. In: *Handbook of Computational Statistics: Data Visualization*. Ed. by C. Chen, W. Härdle, and A. Unwin. Vol. III. Heidelberg: Springer-Verlag, 2006, ???–???

[10] Michael Friendly. “A Brief History of Data Visualization”. en. In: *Handbook of Data Visualization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 15–56. ISBN: 978-3-540-33036-3 978-3-540-33037-0. DOI: 10.1007/978-3-540-33037-0_2.

[11] Marcus D. Hanwell et al. “The Visualization Toolkit (VTK): Rewriting the Rendering Code for Modern Graphics Cards”. en. In: *SoftwareX* 1-2 (Sept. 2015), pp. 9–12. ISSN: 23527110. DOI: 10.1016/j.softx.2015.04.001.

[12] C. Heine et al. “A Survey of Topology-Based Methods in Visualization”. In: *Computer Graphics Forum* 35.3 (June 2016), pp. 643–667. ISSN: 0167-7055. DOI: 10.1111/cgf.12933.

[13] J. D. Hunter. “Matplotlib: A 2D Graphics Environment”. In: *Computing in Science Engineering* 9.3 (May 2007), pp. 90–95. ISSN: 1558-366X. DOI: 10.1109/MCSE.2007.55.

[14] John Hunter and Michael Droettboom. *The Architecture of Open Source Applications (Volume 2): Matplotlib*. https://www.aosabook.org/en/matplotlib.html.

[15] “Jet Bundle”. en. In: *Wikipedia* (Dec. 2020).

[16] John Krygier and Denis Wood. *Making Maps: A Visual Guide to Map Design for GIS*. English. 1 edition. New York: The Guilford Press, Aug. 2005. ISBN: 978-1-59385-200-9.

[17] W A Lea. “A Formalization of Measurement Scale Forms”. en. In: (), p. 44.

[18] *Locally Trivial Fibre Bundle - Encyclopedia of Mathematics*. https://encyclopediaofmath.org/wiki/Locally_trivial

[19] Connie Malamed. *Information Display Tips*. https://understandinggraphics.com/visualizations/information-display-tips/. Blog. Jan. 2010.

[20] *Möbius Strip in nLab*. https://ncatlab.org/nlab/show/M%C3%B6bius+strip.

[21] “Monoid”. en. In: *Wikipedia* (Jan. 2021).

[22] Hans-Georg Müller, Ulrich Stadtmüller, and Fang Yao. "Functional Variance Processes". In: *Journal of the American Statistical Association* 101.475 (2006), pp. 1007–1018.

[23] T Munzner. "Marks and Channels". In: *Visualization Analysis and Design*, pp. 94–114.

[24] Tamara Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC press, Oct. 2014. ISBN: 978-1-4665-0891-0.

[25] Jana Musilová and Stanislav Hronek. "The Calculus of Variations on Jet Bundles as a Universal Approach for a Variational Formulation of Fundamental Physical Theories". In: *Communications in Mathematics* 24.2 (Dec. 2016), pp. 173–193. ISSN: 2336-1298. DOI: 10.1515/cm-2016-0012.

[26] "Quotient Space (Topology)". en. In: *Wikipedia* (Nov. 2020).

[27] James O Ramsay. *Functional Data Analysis*. Wiley Online Library, 2006.

[28] "Retraction (Topology)". en. In: *Wikipedia* (July 2020).

[29] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. "Declarative Interaction Design for Data Visualization". en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. Honolulu Hawaii USA: ACM, Oct. 2014, pp. 669–678. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647360.

[30] "Semigroup Action". en. In: *Wikipedia* (Jan. 2021).

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

[32] David I Spivak. "SIMPLICIAL DATABASES". en. In: (), p. 35.

[33] "Stalk (Sheaf)". en. In: *Wikipedia* (Oct. 2019).

[34] S. S. Stevens. "On the Theory of Scales of Measurement". In: *Science* 103.2684 (1946), pp. 677–680. ISSN: 00368075, 10959203.

[35] Edward R. Tufte. *The Visual Display of Quantitative Information*. English. Cheshire, Conn.: Graphics Press, 2001. ISBN: 0-9613921-4-2 978-0-9613921-4-7 978-1-930824-13-3 1-930824-13-0.

[36] John W Tukey. *Exploratory Data Analysis*. Ed. by Exploratory Data Analysis. Pearson, 1977.

[37] Eric W. Weisstein. *Similarity Transformation*. en. https://mathworld.wolfram.com/SimilarityTransformation.htm Text.

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

[39] Brent A Yorgey. "Monoids: Theme and Variations (Functional Pearl)". en. In: (), p. 12.