# 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

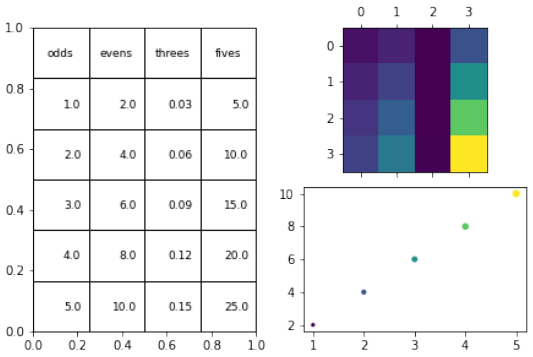



Figure 1: Implicit in visualization is the assumption that these three representations of data are equivalent, specifically that the measurements within a variable and relations of the measurements of each variable are preserved.

# 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 [10] based on these descriptions because the Matplolib artist layer is analogous to the transforms.

## 1.2 What is a viz

? Acquired codes of meaning

# 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

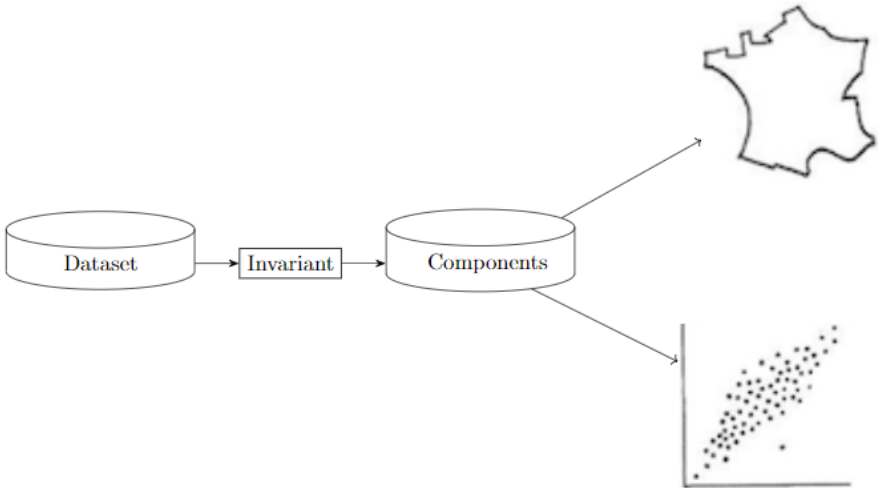



Figure 2: To go from a dataset to a visualization, the data is subset based on a set of constraints (the invariant). The resulting subset becomes the components that are visualized, but the choice of visualization is dependent on the type and structure of the component variables.

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 [**bertin˙semiology˙2011**]. In figure **??**, the *invariant* common to all the data being visualized is "sepal length", "petal length", and "species" and the *components* are

the measurements of these variables. 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

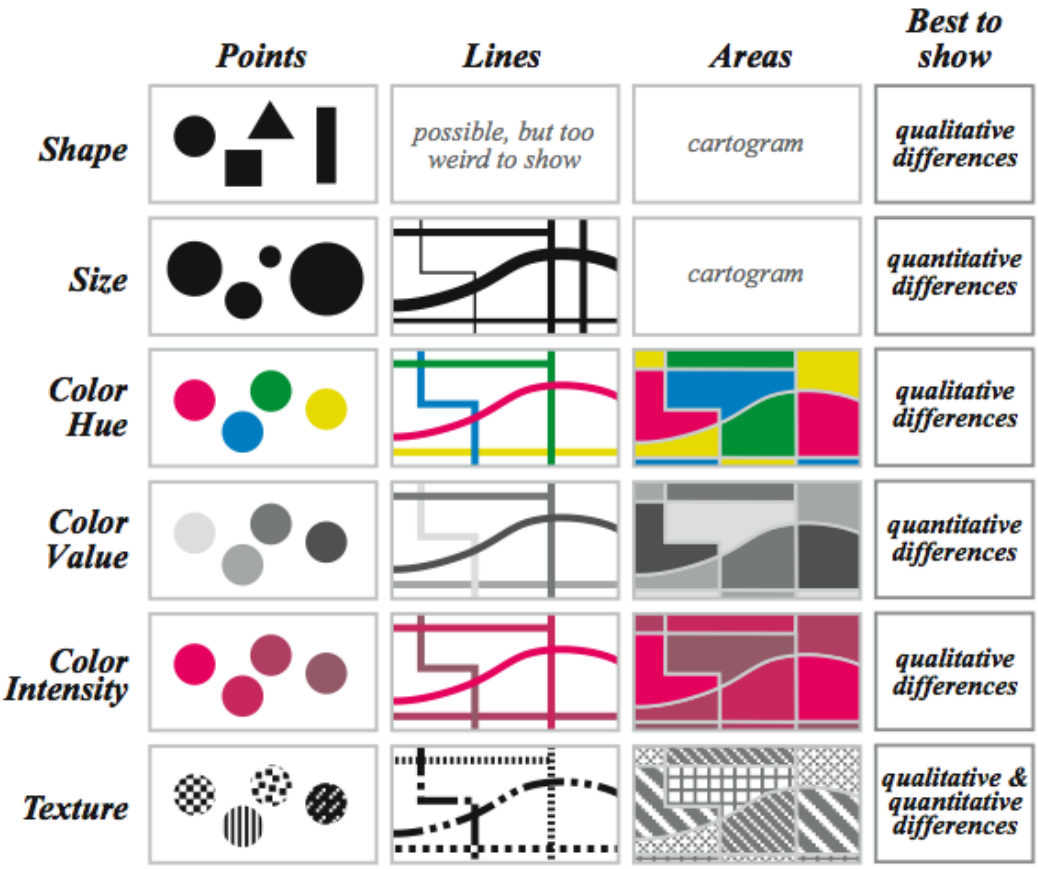



Figure 3: Retinal variables are a codification of how position, size, shape, color and texture are used to illustrate variations in the *components* of a visualization. This tabular form of Bertin's retinal variables is from Understanding Graphics [**˙information˙????**] who reproduced it from *Making Maps: A Visual Guide to Map Design for GIS* [**krygier˙making˙2005**]

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

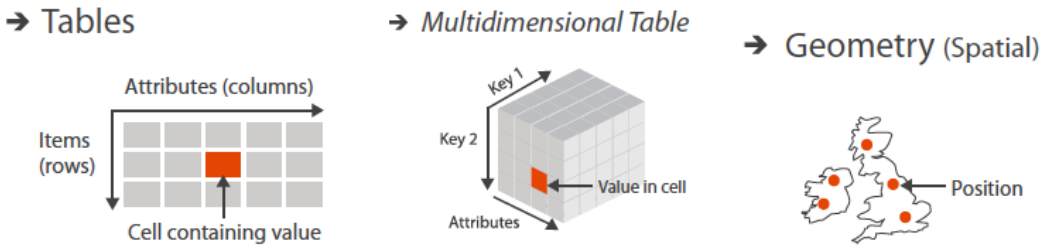



Figure 4: Keys are unique lookup values used to find individual observations in the dataset. Keys are positional references, and can be coordinates on a map or unique values such as a primary key in a database or a (time, latitude, longitude) index in a data cube. Image modified from a diagram from Munzner's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

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 [**munzner˙visualization˙2014**]. 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[**ramsay˙functional˙2006**, **muller˙functional˙2006**] 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.

# 3 Notation & Definitions

In this section 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 use fiber bundles[7, 21] to model data and graphics because they allow us to seperate the variables in a dataset from how the values are connected to each other [3, 4]. The fiber bundles mentioned in this work are assumed to be locally trivial[13, 23].

We first describe the fiber bundle spaces of data(3.1), graphics(3.2), and intermediate visual characteristics (3.3). 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

Lets say we are working with the Global Historical Climatology Network[14], which is a global daily record of weather station such as rainfall and temperature. Depending on

which aspects of the data we want to visualize, we can encode it as a timeseries or map or both or even represent the measurements as completely disconnected.

Using a fiber bundle allows us to encode the properties of the heterogenous variables (3.1.1), the connectivity of the measurements (3.1.3), and functions between these spaces that yield the measurements (3.1.4).

### 3.1.1 Variable Field in F

The fiber is the cartesian product of the variable spaces:

$$F = F_0 \times \ldots \times F_n \tag{2}$$

We formulate the fiber such that it has a mapping between the somethine thing and the field name, as discussed in Spivak's simplacial description of databases [24, 25].

Fibers do not need to be 1D.

### 3.1.2 Measurement Type

An important property of data fields are the measurement type, which we generalize to identifying the left monoid action such that the monoid actions $M$

$$M = M_0 \times \ldots \times M_i \tag{3}$$

is the cartesion cross product of monoids applied to each $F_i \in F$. A monoid [15] $M$ is a set with an associative binary operator $* : M \times M \to M$. A monoid has an identity element $e \in M$ such that $e * a = a * e = a$ for all $a \in M$. A left monoid action [1, 22] of $M$ is a set $F$ with an action $\bullet : M \times F \to F$ with the properties:

**associativity** for all $m, t \in M$ and $x \in F, m \bullet (t \bullet x) = (m \bullet t) \bullet x$

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

The Steven's measurement scales are identified via the group actions on $F_i$ [12, 27] they support. For example, nominal variables are permutable and interval values are translatable.

We identify monoid actions, rather than group, to support an extended variety of scales such as partial order.

### 3.1.3 Connectivity: Base Space

We model data as a fiber bundle ($E$, $K$, $\pi$, $F$)

$$F \hookrightarrow E \xrightarrow{\pi} K \tag{4}$$

with topological total space $E$, base space $K$, fiber space $F$, and the map from total space to base space $\pi : E \to K$. Maps from $K$ to $E$ are called sections and select specific points in $K$. The global space of sections in $E$ is $\Gamma(E)$.

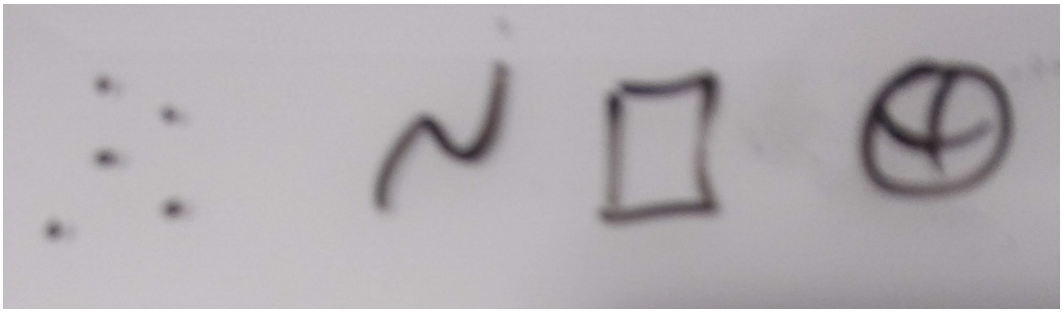


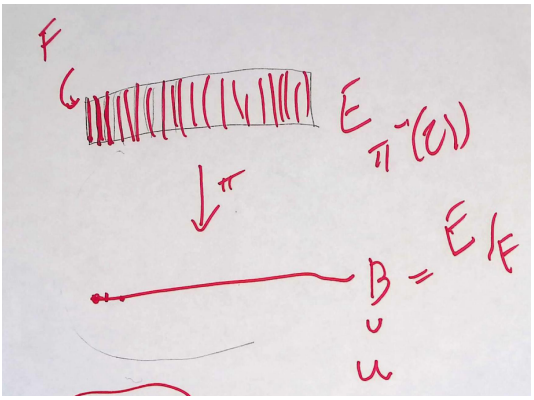
Figure 5: The topological base space $K$ encodes the connectivity of the data space, for example if the data is independent points or a map or on a sphere

Datasets have a semantic topology such as the values are interpreted as discrete observations or part of a timeseries or map, or nodes in a networks [9, 17].

As illustrated in fig **??**, $K$ is this underlying structure. $K$ does not directly know the values; instead it is the sections that define the lookup between keys $k \in K$ and the corresponding values in $E$.

The topology $K$ and the

$F$ are determined by how $E$ is subdivided[19]. In figure **??** we can divide a rectangular base space such that there is a short fiber and long base space or a long fiber and short base space. This is analogous to long and wide forms of the same table [29].



### 3.1.4 Data: Section

The sections of the fiber bundles are the instances of the data. They are the functions that map from a point $k \in K$ to a set of values in $F$. For example, if we have a database, the section is the table that yields a row given an index. The section $\tau$ is the mapping from base space to total space $\tau : K \to E$

$$\begin{array}{ccc} F & \hookrightarrow & E \\ & & \pi \downarrow \uparrow \tau \\ & & K \end{array} \tag{5}$$

such that in a trivial fiber bundle, $E = K \times F$ [7, 21]:

$$\tau(k) = (k, (x_{F_0}, \ldots, x_{F_n})) \tag{6}$$

which we can also decompose such that

$$\tau_0(k) = (k, (x_{F_0})) \tag{7}$$

$$\vdots \tag{8}$$

$$\tau_n(k) = (k, (x_{F_n})) \tag{9}$$

which allows us to have field wise access to the data so long as there is a shared $K$.

### 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 $\mathcal{O}(E)$

$$\begin{array}{ccc} \iota^* E & \xleftarrow{\iota^*} & E \\ {\scriptstyle\pi}\downarrow\uparrow{\scriptstyle\iota^*\tau} & & {\scriptstyle\pi}\downarrow\uparrow{\scriptstyle\tau} \\ U & \xleftarrow{\iota} & K \end{array} \tag{10}$$

which is the localized section of fibers $\iota^*\tau : U \to \iota^* E$ pulled back 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 $\mathcal{F}_b$ [23, 26].

The jet bundle $\mathcal{J}$ [11, 18] is a for writing differential equations on sections of fiber bundles; this information is required for some visual characteristics, such as line thickness.

### 3.1.6 Example: Temperature

Moved & walked through b/c was getting clunky to not have terms yet

The fiber bundle model is flexible enough to express some of the many different forms that temperature data can come in.

The datasets in figure **??** have identical fibers that encode a set of temperature values. In figure **??** the temperatures lie on a line such that a section could return a timeseries or a distribution. In figure **??**, the temperatures lie on a 2D continuous plane; a section could return a map or contour. Because the fiber is 1D, it does not encode metadata

To encode the metadata, the fiber is expanded as illustrated in figure **??**. The fiber in figure **??** is the cartesian product of the space of possible temperature values in degrees celsius and space of possible time values

$$F = [temp_{min}, temp_{max}] \times [time_{min}, time_{max}] \tag{11}$$

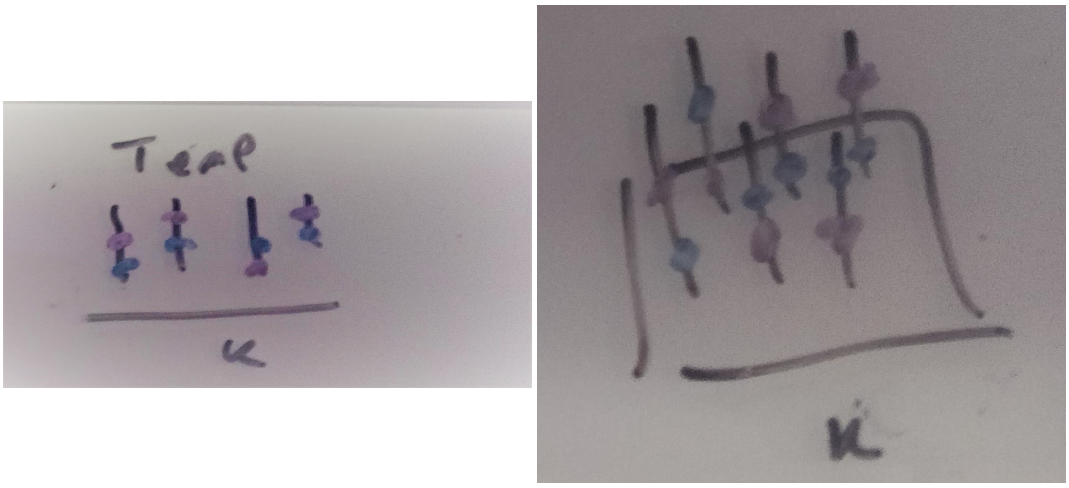



Figure 6: These two datasets have the same fiber of temperature but different base spaces. In figure **??** the temperature values are 1D continuous, while in figure **??** the temperature values are 2D continuous.

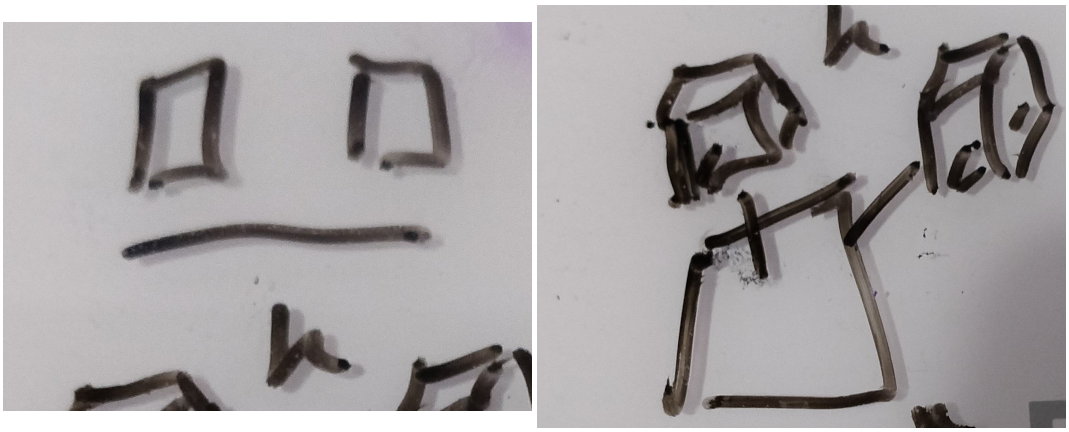


Figure 7: The fiber is expanded to include metadata fields that describe the semantics of $K$. In figure **??** the fiber is temperature $\times$ time and in figure **??** the fiber is temperature$\times$ latitude $\times$ longitude
.

while the fiber in figure **??** is the cartesian product of temperature, latitude, and longitude

$$F = [temp_{min}, temp_{max}] \times [-90, 90] \times [-180, 180] \tag{12}$$

such that $E$ is the space of all possible points in $F$.

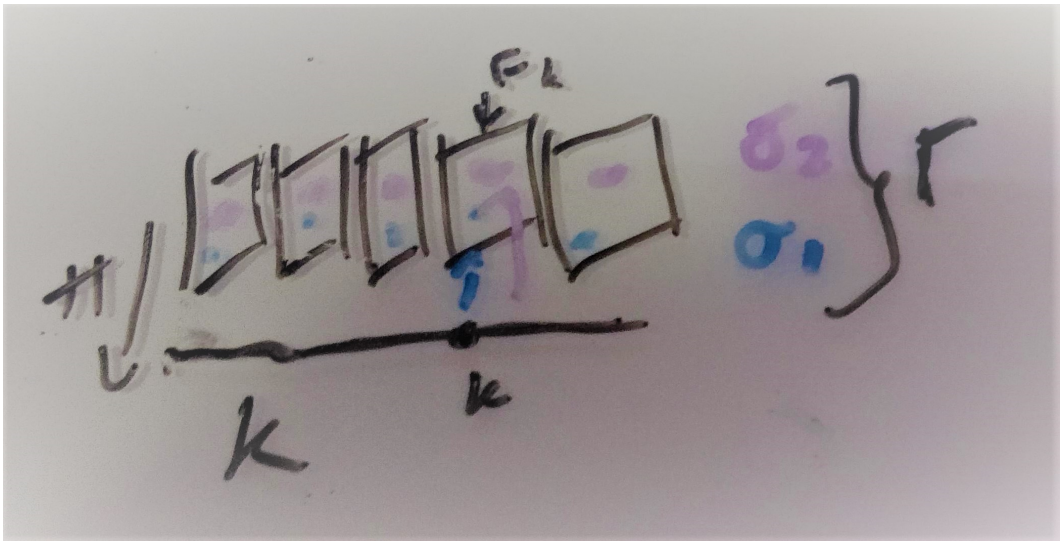


Figure 8: The section $\tau_1$ returns the blue points, while $\tau_2$ returns the purple points. $\Gamma(E)$ is the set of all sections, including $\tau_1$ and $\tau_2$
.

Given the fiber described in equation 11, the sections $\tau_1$ and $\tau_2$ in figure **??** return tuples of the form

$$\tau(k) = (k, (temperature, time)) \tag{13}$$

such that sections with the constraint that time is monotonic return a timeseries.

## 3.2 Prerender Space $H$

We define a graphic space $H$ such that we do not have to assume the physical output space of the renderer. This means that the graphic in $H$ can be output to a screen or 3D printed space or a dome. We model the prerender space as a fiber bundle (H, S, $\pi$, D). $H$ is the predisplay space, with a fiber $D$ dependent on the target display and a base space of $S$.

### 3.2.1 Base space

The underlying topology $S$ of a graphic often needs more dimensions than the data topology $K$ because of the specifications of the display space. For example, a line plot on a plane

(such as a screen or a piece of paper) by necessity needs to also have a thickness so that it is visible, which maps back to a set of connected points in $H$. The topology of these connected points is therefore the region $s \subset S$ such that $\xi : S \to K$ is a deformation retraction [20]

$$\begin{array}{ccc} E & & H \\ {\scriptstyle\pi}\downarrow & & {\scriptstyle\pi}\downarrow \\ K & \xleftarrow{\xi} & S \end{array} \tag{14}$$

that goes from a region $s \in S_k$ to its associated point $k$, such that when $\xi(s) = k$, $\xi^* \tau(s) = \tau(k)$.

### 3.2.2 Fiber and Section

A section $\rho : S \to H$ is a mapping from a region $s$ on a mathematical encoding of the image to a region $xy$ on the screen that the renderer then maps to visual space as defined in D.

**Example** For a physical screen display, we can consider a predisplay space that is a trivial fiber bundle $H = \mathbb{R}^5 \times S$ such that $\rho$ is

$$\rho(s) = [x(s), y(s), r(s), g(s), b(s)] \tag{15}$$

To draw an image, a region, $H$ is inverse mapped into a region $s \in S$ where

$$s = \rho_{XY}^{-1}(xy) \tag{16}$$

such that the rest of the fields in $\mathbb{R}^7$ are then integrated over $s$ to yield the remaining fields:

$$r = \iint_s \rho_R(s) ds^2 \tag{17}$$

$$g = \iint_s \rho_G(s) ds^2 \tag{18}$$

$$b = \iint_s \rho_B(s) ds^2 \tag{19}$$

Here we assume a single opaque 2D image such that the $z$ and *alpha* fields can be omitted. To support overplotting and transparency, we can consider $D = R^7$

### 3.2.3 Example

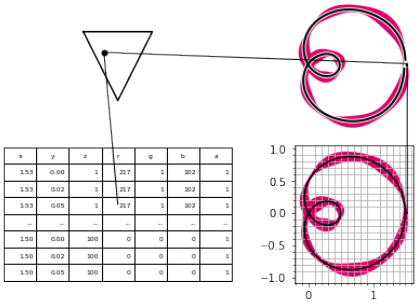



Figure 9

As illustrated in figure 9, words.

## 3.3 Artist

In this section we will define the artist as a mapping from a sheaf $\mathcal{O}(E)$ to $\mathcal{O}(H)$.

$$A : \mathcal{O}(E) \to \mathcal{O}(H) \tag{20}$$

The artist decomposes to mapping data to visual $\nu : E \to V$, then compositing $V$ pulled back along $\xi$ to $\xi^* V$ to a visual mark in prerender space $Q : \xi^* V \to H$.

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

The pullback map $\xi^*$ copies each value in $V$ over $k$ to $s$ in corresponding $S_k$ such that $\xi^* V$ can have multiple values that map to one value in $V$.

The visual fiber bundle $(V,\ K,\ \pi,\ P)$ has section $\mu : V \to K$ that resolves to a visual variable [2, 16] in fiber $P$. The visual transformer $\nu$ is a set of functions each targeting a

different $\mu$

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

where $\mu_i$ are the visual parameters in the assembly function $Q(\mu_0, \ldots, \mu_n)(s) = \rho(s)$.

### 3.3.1 Example: Matplotlib Visual Fiber

For example, for Matplotlib [10], some of the possible types in $P$ are:

| $\boldsymbol{\nu_i}$ | $\boldsymbol{\mu_i}$ | $\boldsymbol{codomain(\nu_i)}$ |
|---|---|---|
| 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 **??** is an example of the visual fiber defined in terms of common parameters to plots in Matplotlib. The range of $\mu_i$ determine the monoid actions on $\mu_i$. A section of V $\mu$ is a tuple of visual values that specifies the visual characteristics of a glyph. For example, given a fiber of $\{xpos, ypos, color\}$ one section is $\{.5, .5, (255, 20, 147)\}$. $Q$ determines how this section is applied to a graphic.

### 3.3.2 Visual Channels

$\nu : E \to V$ is an equivariant map such that there is a homomorphism from left monoid actions on $E_i$ to left monoid actions on $V_i$ where $i$ identifies a field in the fiber. $E_i$ and $V_i$ each contain a set of values as defined in $F$ and $P$ respectively. A validly constructed $\nu$ is one where the diagram

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

commutes such that $\nu_i(m_e(E_i)) = m_v(\nu_i(E_i))$.

**Example: Ordering** To preserve ordering of elements in $E_i$, $\nu$ must be a monotonic function such that given $e_1, e_2 \in E_i$

$$\text{if } e_1 \leq e_2 \text{ then } \nu(e_1) \leq \nu(e_2) \tag{24}$$

**Example: Translation** According to Stevens, interval data is a set with general linear group actions [12, 27]. Position is a visual variable that can support translation

$$\nu(x + c) = \nu(x) + \nu(c) \tag{25}$$

**Example: Invalid $\nu$** Given a transform $t(x) = x + 2$, we construct a $\nu$ that always takes data to .5:

$$\begin{array}{ccc} E_1 & \xrightarrow{\lambda : e \mapsto .5} & V_i \\ {\scriptstyle 2e}\downarrow & & \downarrow{\scriptstyle 2v} \\ E_1 & \xrightarrow{\lambda} & V_1 \end{array} \tag{26}$$

This $\nu$ is invalid because the graph does not commute for $t$:

$$\nu(t(e)) \stackrel{?}{=} t(\nu(e)) \tag{27}$$

$$.5 \stackrel{?}{=} t(.5) \tag{28}$$

$$.5 \neq 2 * .5 \tag{29}$$

To construct a valid $\nu$, the diagram must commute for all monoid actions on the sets in $E_i, V_i$.

### 3.3.3 Assembling Marks

As shown in figure **??**, $Q$ takes the individual fields in $V$ as input and outputs a single piece of a graphic on $H$. As with $\nu$, the constraint on $Q$ is that for every monoid actions on the

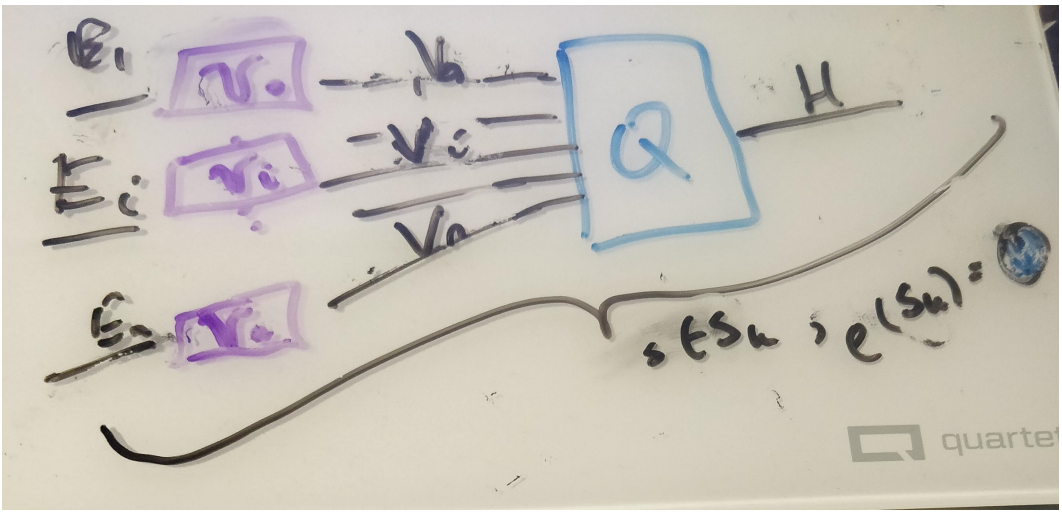



Figure 10: The $\nu$ functions convert data $E$ to visual $V$. $Q$ assembles the different types of visual parameters $V_i$ into a graphic in $H$. $Q \circ \mu(\xi^{-1}J)$ forms a visual mark by applying $Q$ to a region mapped to connected components $J \subset K$.

input in $V$ there is a corresponding monoid action on the output in $H$

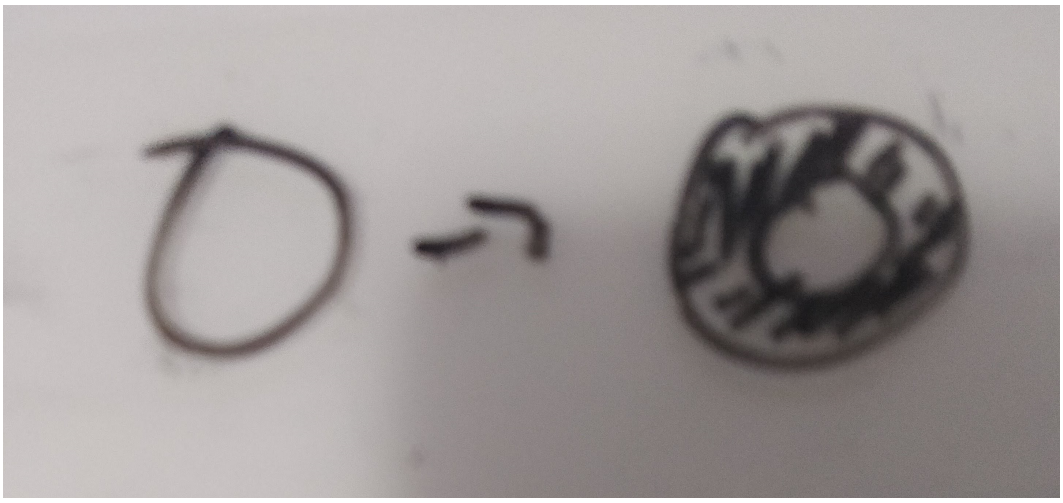
$$Q : \Gamma(V) \to \Gamma(H) \tag{30}$$

When $Q : \mu \mapsto \rho$ yields a $\rho$ that maps to the same values in $D$ over all $S_k$, then $M$ can be defined over $\Gamma(H)$ such that a constraint on $Q$ is that it must be equivariant. For example, when $\mu_i$ is the color of the glyph, it maps directly to (R,G,B) in D.

Many $\mu_i$ are graphical parameters that do not apply to the whole glyph, such as edge thickness in figure **??**.

In these situations, not all $\rho$ in $\Gamma(H)$ will support these parameters; instead we define an action on the output graphic $Q(\Gamma(V)) \in \Gamma(H)$ since by definition every section $\mu$ will have a corresponding $\rho$.

We then define the constraint on $Q$ such that if $Q$ applied to two sections $\mu, \mu\prime$ generate the same graphic $\rho$, then the output of both sections acted on by the same monoid $m$ must also be the same.



Lets call the visual encodings $\Gamma(V) = X$ and the graphic $Q(\Gamma(V)) = Y$. If $\forall m \in M$ and $\forall \mu, \mu' \in X$,

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \tag{31}$$

then a group action on $Y$ can be defined as $m \circ \rho = \rho'$ where $\rho' = Q(g \circ \mu)$ with $\mu \in Q^{-1}(\rho)$.

Given

- $P = \{xpos, ypos, color, thickness\}$
- $\mu = 0, 0, 0, 1$
- $Q(\mu) = \rho$ generates a piece of the thin circle in figure **??**

the constraint on $Q$ means that the translation $m = \{e, e, e, x + 2\}$ applied to $\mu$ such that $\mu' = \{0, 0, 0, 3\}$ has an equivalent action on $\rho$ that causes $Q(\mu\prime)$ to be equivalent to the thicker circle in figure **??**.

**Example: Invalid Q** Insert some degenerate Q that generates an inconsistent glyph

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

constraint: inputs go to same o utput means changes to inputs mean same changes to output

**Graphical Marks** To output a mark [2, 5], $Q$ is called with all the regions $s$ that map back to a set of connected components $J \subset K$:

$$J = \{j \in K \text{ s. t. } \exists \gamma \text{ s.t. } \gamma(0) = k \text{ and } \gamma(1) = j\} \tag{32}$$

where the path[6] $\gamma$ from $k$ to $j$ is a continous function from the interval [0,1].

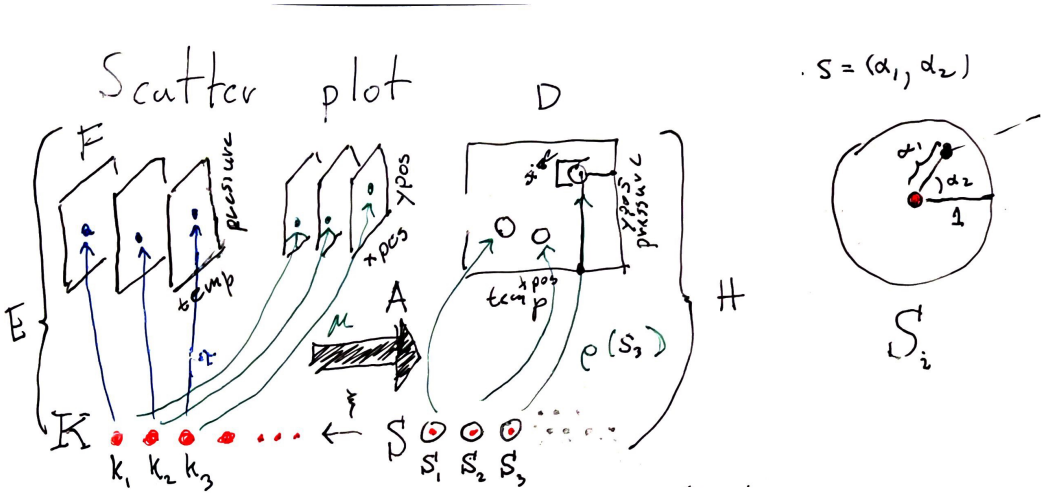
We define the mark as the graphic generated by $Q(S_j)$

$$H \underset{\rho(S_j)}{\overset{}{\rightleftarrows}} S_j \underset{\xi^{-1}(J)}{\overset{\xi(s)}{\rightleftarrows}} J_k \tag{33}$$

in terms of $K$ because mark is a semantic term denoting the graphic representation of the data.

### 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 simple heatmap.




**Q: scatter plot**

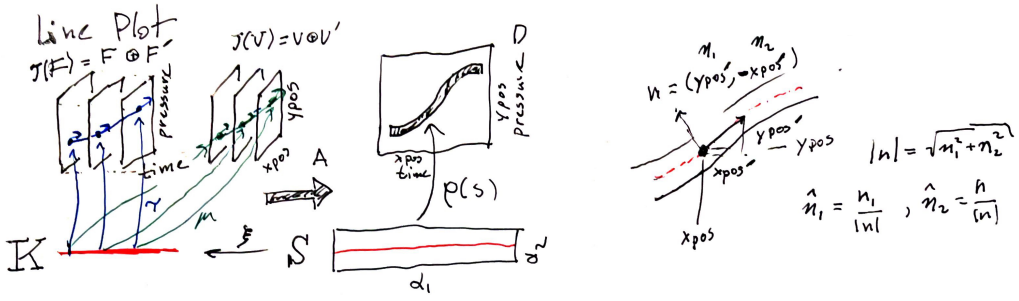
$$Q(xpos, ypos)(\alpha_1, \alpha_2) \tag{34}$$

Given a defaut color of black, $\rho_{RGB} = (0, 0, 0)$. The position of this swatch of color can be computed relative to the location on the disc $S_i$ as shown in figure **??**:

$$x = size \bullet \alpha_1 \bullet \cos(\alpha_2) + xpos \tag{35}$$

$$y = size \bullet \alpha_1 \bullet \sin(\alpha_2) + ypos \tag{36}$$

such that $\rho(s) = (x, y, 0, 0, 0)$ where $s$ is the region in $H$.




**Q: line plot** The line plot shown in fig **??** exemplifies the need for the jet discussed in section **??**

$$Q(xpos, \hat{n_1}, ypos, \hat{n_2})(\alpha_1, \alpha_2) \tag{37}$$

where the magnitude of the thickness is

$$|n| = \sqrt{{n_1}^2 + {n_2}^2} \tag{38}$$

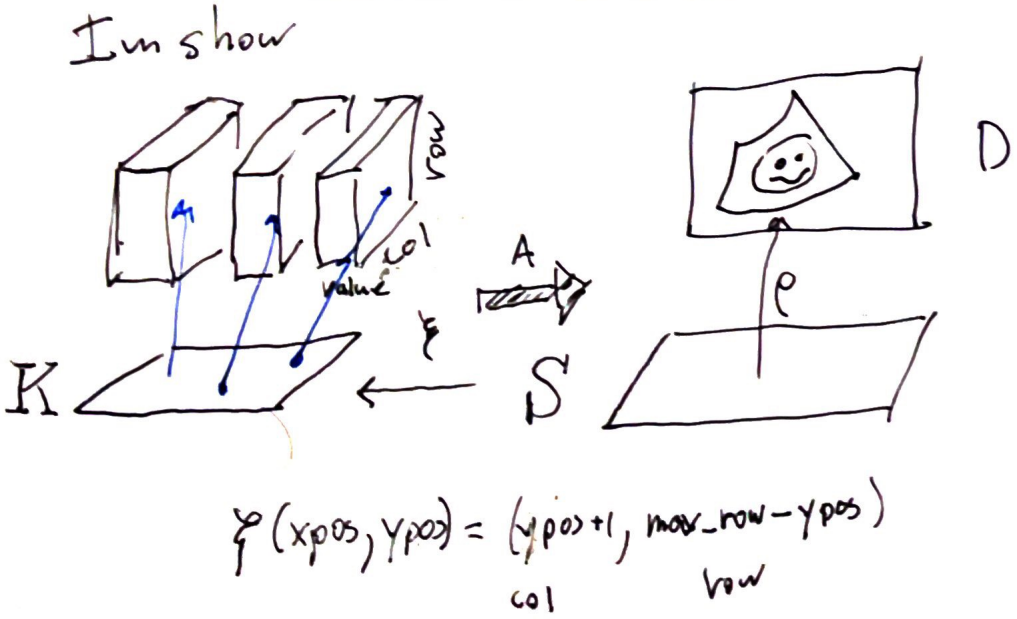
such that the components are

$$\hat{n_1} = \frac{n_1}{|n|}, \ \hat{n_2} = \frac{n_2}{|n|} \tag{39}$$

which yields components of $\rho(s)$:

$$x = xpos(\xi(\alpha_1)) + \alpha_2\hat{(}n_1)(\xi(\alpha_1)) \tag{40}$$

$$y = ypos(\xi(\alpha_1)) + \alpha_2\hat{(}n_2)(\xi(\alpha_2)) \tag{41}$$




**Q: heatmap** The heatmap in figure **??**

$$Q(xpos, ypos, color) \tag{42}$$

has in the simple case a direct lookup into $K$ to obtain the $\mu = (x, y, c)$ values that are mapped into

$$D_{RGB} = color(\xi(\alpha_1, \alpha_2))D_x \qquad = xpos(\xi(\alpha_1, \alpha_2))D_y = ypos(\xi(\alpha_1, \alpha_2)) \tag{43}$$

through $\rho$.

### 3.4 Making the fiber bundle computable

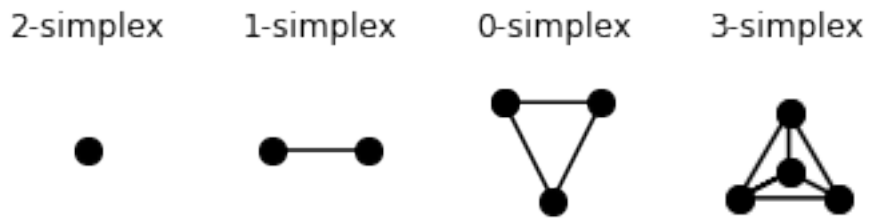
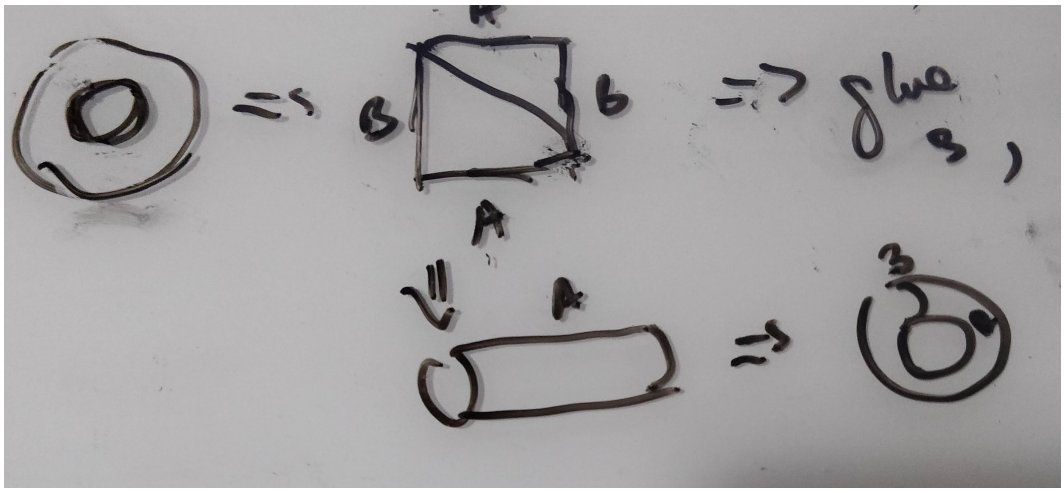



Figure 11: Simplices can encode the connectivity of the data, from fully disconnected (0 simplex) observations to all observations are connected to at least 3 other observations

One way to build flexible $\xi$ is to choose a consistent way of representing$K$. In our draft implementation of the data as fiber bundle model, we triangularize $K$ using complexes of the simplicies shown in figure 11 such that $\xi$ consistently yields some combination of vertexes, edges, and faces. This gives a common data indexing structure on which to build components that could potentially be reused across $Q$.



**Example** Given data that lies on the toroidal space shown in figure **??**, the torus $K$ can be implemented as a simplacial complex of two 2-simplexes. We unfold the torus into the two triangles that compose the square in figure **??** and encode that as K. We also

constrain the functions on $A$, $A'$, $B$ and $B'$ such that $A$ can be glued to $A'$ and $B$ to $B'$ to reconstruct the torus.

### 3.4.1 Visual Idioms: Equivalance class of artists

As formulated above, every artist function $A$ has fixed $\nu$ and generates a distinct graphic $\rho$. It is unfeasible to implement $A$ for every single graphic; instead we implement the equivalence class of artists $\{A \in A' : A_1 \equiv A_2\}$ which is $Q : \Gamma(V) \to \Gamma(H)$.

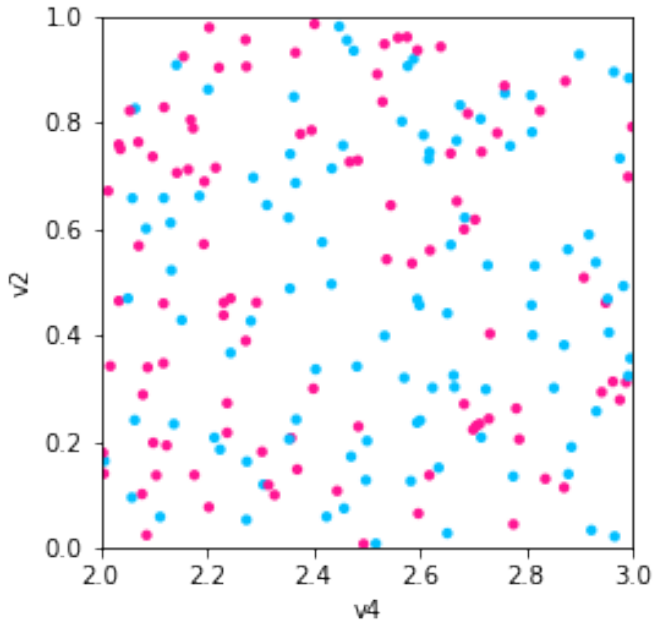
# 4 Matplottoy

We build on the existing Matplotlib architecture [10] 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 [8, 28] 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

$$Q(\mu_{xpos}, \mu_{ypos}, \mu_{facecolor}, \mu_s) \tag{44}$$

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 44, 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 $\mu$ 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 $\mu$ and whether the $\nu$ functions are valid for the input data.

- assume that the $\nu$ 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?

```
1  def __init__(self, data, transforms, *args, **kwargs):
2      super().__init__(*args, **kwargs)
3      # check that the data you're trying to transform
4      # has a way to provide vertex data
5      assert 'vertex' in data.FB.K['tables'] #there is no check here
6
7      # check that you've given the required parameters
8      utils.check_constraints(Point, transforms.keys())
9      utils.validate_transforms(data.FB.F, transforms)
10
11     self.data = data
12     self.transforms = transforms
```

Lines 9-10 attach the data and transforms to the *Point* object as a concession to the Matplotlib architecture that seperates drawing from creation. In `draw`, the attributes of the 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)
```

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

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

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

where the $\nu$ functions are represented as classes so that parameters can be curried. For example, `color.Categorical` is a whole set of $\nu$ 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] 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.

[3] 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.

[4] 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.

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

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

[7] "Fiber Bundle". en. In: *Wikipedia* (May 2020).

[8] 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, ???–???

[9] Berk Geveci et al. "VTK". In: *The Architecture of Open Source Applications* 1 (2012), pp. 387–402.

[10] 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.

[11] "Jet Bundle". en. In: *Wikipedia* (Dec. 2020).

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

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

[14] Matthew J. Menne et al. “An Overview of the Global Historical Climatology Network-Daily Database”. In: *Journal of Atmospheric and Oceanic Technology* 29.7 (July 2012), pp. 897–910. DOI: 10.1175/JTECH-D-11-00103.1.

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

[16] T Munzner. “Marks and Channels”. In: *Visualization Analysis and Design*, pp. 94–114.

[17] Tamara Munzner. “What: Data Abstraction”. In: *Visualization Analysis and Design*. AK Peters Visualization Series. A K Peters/CRC Press, Oct. 2014, pp. 20–40. ISBN: 978-1-4665-0891-0. DOI: 10.1201/b17511-3.

[18] 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.

[19] “Quotient Space (Topology)”. en. In: *Wikipedia* (Nov. 2020).

[20] “Retraction (Topology)”. en. In: *Wikipedia* (July 2020).

[21] Todd Rowland. *Fiber Bundle*. en. https://mathworld.wolfram.com/FiberBundle.html. Text.

[22] “Semigroup Action”. en. In: *Wikipedia* (Jan. 2021).

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

[24] David I Spivak. *Databases Are Categories*. en. Slides. June 2010.

[25] David I Spivak. “SIMPLICIAL DATABASES”. en. In: (), p. 35.

[26] “Stalk (Sheaf)”. en. In: *Wikipedia* (Oct. 2019).

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

[28] John W Tukey. *Exploratory Data Analysis (Specific Chapter_)*. Ed. by Exploratory Data Analysis. Pearson, 1977.

[29] Hadley Wickham et al. "Tidy Data". In: *Journal of Statistical Software* 59.10 (2014), pp. 1–23.