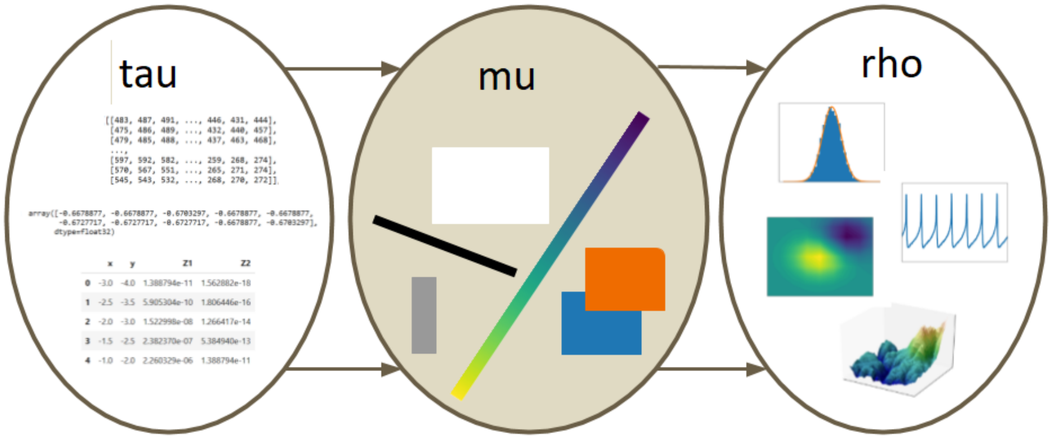
1 Introduction



```
[[483, 487, 491, ..., 446, 431, 444],
 [475, 486, 489, ..., 432, 440, 457],
 [479, 485, 488, ..., 437, 463, 468],
 ...,
 [597, 592, 582, ..., 259, 268, 274],
 [570, 567, 551, ..., 265, 271, 274],
 [545, 543, 532, ..., 268, 270, 272]]
```

```
array([-0.6678877, -0.6678877, -0.6703297, -0.6678877, -0.6678877,
       -0.6727717, -0.6727717, -0.6727717, -0.6678877, -0.6703297],
      dtype=float32)
```

	x	y	Z1	Z2
0	-3.0	-4.0	1.388794e-11	1.562882e-18
1	-2.5	-3.5	5.905304e-10	1.806446e-16
2	-2.0	-3.0	1.522998e-08	1.266417e-14
3	-1.5	-2.5	2.382370e-07	5.384940e-13
4	-1.0	-2.0	2.260329e-06	1.388794e-11

Figure 1: Visualization is equivariant maps between data and visual encoding of the variables and assembly of those encodings into a graphic. will replace w/ overarching figure w/ same structure

While many researchers have identified and described important aspects of visualization, they have specialized in such different ways as to not provide a model general enough to natively support the full range of data and visualization types many general purpose modern visualization tools may need to support. The work presented in this paper is motivated by a need for a visualization library that developers could build complex, domain specific tools tuned to the semantics and structure carried in domain specific data. The core architecture also needs to be robust to the big data needs of many visualization practitioners, and therefore support distributed and streaming data needs. To support both exploratory and confirmatory visualization[47], this tool needs to support 2D and 3D, static, dynamic and interactive visualizations.

Specifically, this work was driven by a rearchitecture of the Python visualization library Matplotlib[24] to meet modern data visualization needs. We aim to take advantage of developments in software design, data structures, and visualization to improve the consistency, composibility, and discoverability of the API. To do so, this work first presents a mathematical description of how data is transformed into graphic representations, as shown in figure 1. As with other mathematical formalisms of visualization [25, 28, 42, 49], a mathematical framework provides a way to formalize the properties and structure of the visualization. In contrast to the other formalisms, the model presented here is focused on the components that build a visualization rather than the visualization itself.

In other words this model is not intended to be evaluative, it is intended to be a reference specification for visualization library API. To make this model as implementation independent as possible, we propose fairly general mathematical abstractions of the data container such that we do not need to assume the data has any specific structure, such as a relational database. We reuse this structure for the graphic as that allows us to specifically discuss how structure is preserved. We take a functional approach because functional paradigms encourage writing APIs that are flexible, concise and predictable due to the lack of side effects [27]. Furthermore, by structuring the API in terms of composition of the smallest units of transformation for which we can define correctness, a functional paradigm naturally leads to a library of highly modular components that are composable in such a way that by definition the composition is also correct. This allows us to ensure that domain specific visualizations built on top of these components are also correct without needing knowledge of the domain. As with the other mathematical formalisms of visualization, we factor out the rendering into a separate stage; but, our framework describes how these rendering instructions are generated.

In this work, we present a framework for understanding visualization as equivariant maps between topological spaces. Using this mathematical formalism, we can interpret and extend prior work and also develop new tools. We validate our model by using it to re-design artist and data access layer of Matplotlib, a general purpose visualization tool.

2 Background

One of the reasons we developed a new formalism rather than adopting the architecture of an existing library is that most information visualization software design patterns, as categorized by Heer and Agrawala[22], are tuned to very specific data structures. This in turn restricts the design space of visual algorithms that display information (the visualization types the library supports) since the algorithms are designed such that the structure of data is assumed, as described in Tory and Möller's taxonomy [45]. In proposing a new architecture, we contrast the trade offs libraries make, describe different types of data continuity, and discuss metrics by which a visualization library is traditionally evaluated.

2.1 Tools

One extensive family of relational table based libraries are those based on Wilkenson's Grammar of Graphics (GoG) [52], including ggplot[51], protovis[7] and D3 [8], vega[36] and altair[48]. The restriction to tables in turn restricts the native design space to visualizations suited to tables. Since the data space and graphic space is very well defined in this grammar, it lends itself to a declarative interface [23]. This grammar oriented approach allows users to describe how to compose visual elements into a graphical design [53], while we are proposing a framework for building those elements. An example of this distinction is that the GoG grammar includes computation and aggregation of the table as part of the grammar, while we propose that most computations are specific to domains and only try to describe them when they are specifically part of the visual encoding - for example mapping data to a color. Disentangling the computation from the visual transforms allows us to determine whether the visualization library needs to handle them or if they can be more efficiently computed by the data container.

A different class of user facing tools are those that support images, such as ImageJ[37] or Napari[38]. These tools often have some support for visualizing non image components of a complex data set, but mostly in service to the image being visualized. These tools are ill suited for general purpose libraries that need to support data other than images because the architecture is oriented towards building plugins into the existing system [54] where the image is the core data structure. Even the digital humanities oriented ImageJ macro ImagePlot[41], which supports some non-image aggregate reporting charts, is still built around image data as the primary input.

There are also visualization tools where there is no single core structure, and instead internally carry around many different representations of data. Matplotlib, has this structure, as does VTK [20, 21] and its derivatives such as MayaVi[35] and extensions such as ParaView[4] and the infoviz themed Titan[9]. Where GoG and ImageJ type libraries have very consistent APIs for their visualization tools because the data structure is the same, the APIs for visualizations in VTK and Matplotlib are significantly dependent on the structure of the data it expects. This in turn means that every new type of visualization must carry implicit assumptions about data structure in how it interfaces with the input data. This has lead to poor API consistency and brittle code as every visualization type has a very different point of view on how the data is structured. This API choice particularly breaks down when the same dataset is fed into visualizations with different assumptions about structure or into a dashboard consisting of different types of visualization[1, 18] because there is no consistent way to update the data and therefore no consistent way of guaranteeing that the views stay in sync. Our model is a structure dependent formalism, but then also provides a core representation of that structure that is abstract enough to provide a common interface for many different types of visualization.

2.2 Data

Discrete and continuous data and their attributes form a discipline independent design space [34], so one of the drivers of this work was to facilitate building libraries that could natively support domain specific data containers that do not make assumptions about data continuity. Fiber bundles are one such way model to model containers, as Butler proposes because they encode the continuity of the data separately from the types of variables and are flexible enough to support discrete and ND continuous datasets [10, 11]. Since Butler's model lacks a robust way of describing the variables, we fold in Spivak's Simplacial formulation of databases [39, 40] so that we can encode a schema like description of the data in the fiber bundle.

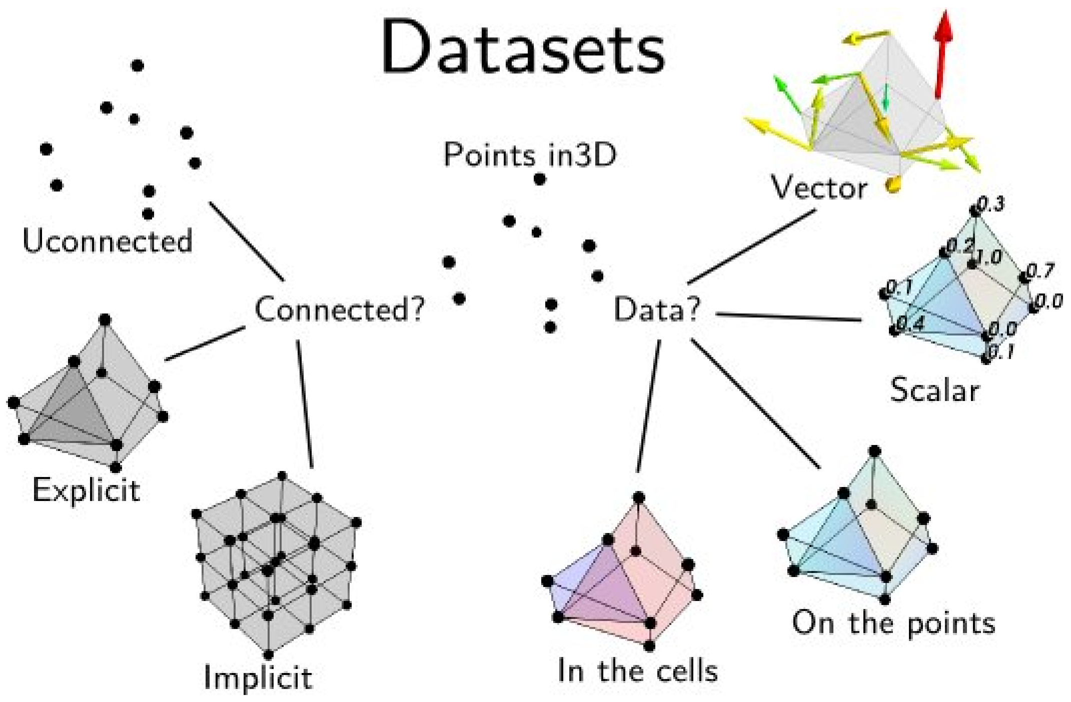


Figure 2: One way to describe data is by the connectivity of the points in the dataset. A database for example is often discrete unconnected points, while an image is an implicitely connected 2D grid. This image is from the Data Representation chapter of the MayaVi 4.7.2 documentation.[16]

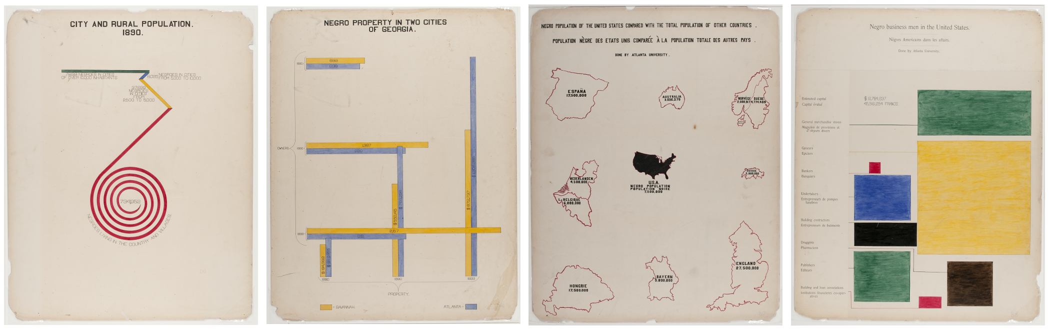
As shown in figure 2, there are many types of connectivity. A database typically consists of unconnected records, while an image is an implicit 2D grid and a network is some sort of explicitly connected graph. In this work we will refer to the points of the dataset as *records* to indicate that a point can be a vector of heterogenous elements. Each *component* of the record is a single object, such as a temperature measurement, a color value, or an image. We also generalize *component* to mean all objects in the dataset of a given type, such as all temperatures or colors or images. The way in which these records are connected is the *connectivity*, *continuity*, or more generally *topology*.

definitions	
records	points, observations, entries
components	variables, attributes, fields
connectivity	how the records are connected to each other

Often this topology has metadata associated with it, describing for example when or where the measurement was taken. Building on the idea of metadata as *keys* and their associated *value* proposed by Munzner [31], we propose that information rich metadata are part of the components and instead the values are keyed on coordinate free structural ids. In contrast to Munzner's model where the semantic meaning of the key is tightly coupled to the position of the value in the dataset, our model allows for renaming all the metadata, for example changing the coordinate systems or time resolution, without imposing new semantics on the underlying structure.

2.3 Visualization

A visualization tool produces a graphical design and an image rendered based on that design, as described by Mackinlay [29]. He defines the graphical design as the set of encoding relations from data to visual representation[28], and the design rendered in an idealized abstract space is what throughout this paper we will refer to as a graphic. In addition to the graphic representations, Byrne et al. describe how visualizations have figurative representations that have meaning due to their similarity in shape to external concepts [12]. Mackinlay proposes that a visualization tool's expressiveness is a measure of how much of the structure of the data the tool encodes, while the tool's effectiveness describes how much design choices are made in deference to perceptual saliency [13–15, 32].



(a) (b) (c) (d)

Figure 3: Du Bois' data portraits[17] of post reconstruction Black American life exemplify that the fundemental characteristics of data visualization is that the visual elements vary in proportion to the source data. In figure 3a, the length of each segment maps to population; in figure 3b, the bar charts are intersected to show the number of property owners and how much they own in a given year; in figure 3c the countries are scaled to population size; and figure 3d is a treemap where the area of the rectangle is representative of the number of businesses in each field. The images here are from the Prints and Photographs collection of the Library of Congress [2, 3, 43, 44]

We propose that the Du Bois visualizations in figure 3 are representative of the expressivity a core architecture should allow a downstream library to express. This is because while the Du Bois figures are not the common scatter, bar, or line plot [19], they conform to the constraint that a visualization is a mapping from data to visuals. In figure 3c, Du Bois combines a graphical representation where glyph size varies by population with a figurative representation of those glyphs as the countries the data is from, which means that the semantic and numerical properties of the data are preserved in the graph. The visual representations in the Du Bois figures are in proportion to the quantitative data being represented, so the a chart is faithful according to Tufte's Lie Principal[46]. The properties of the representation match the properties of the information being represented, so they are fairly understandable according to Norman's Naturalness Principal[33]. As with the Du

Bois data portraits, it is fundamental that any architecture tool we build ensures that the data properties match the visual properties.

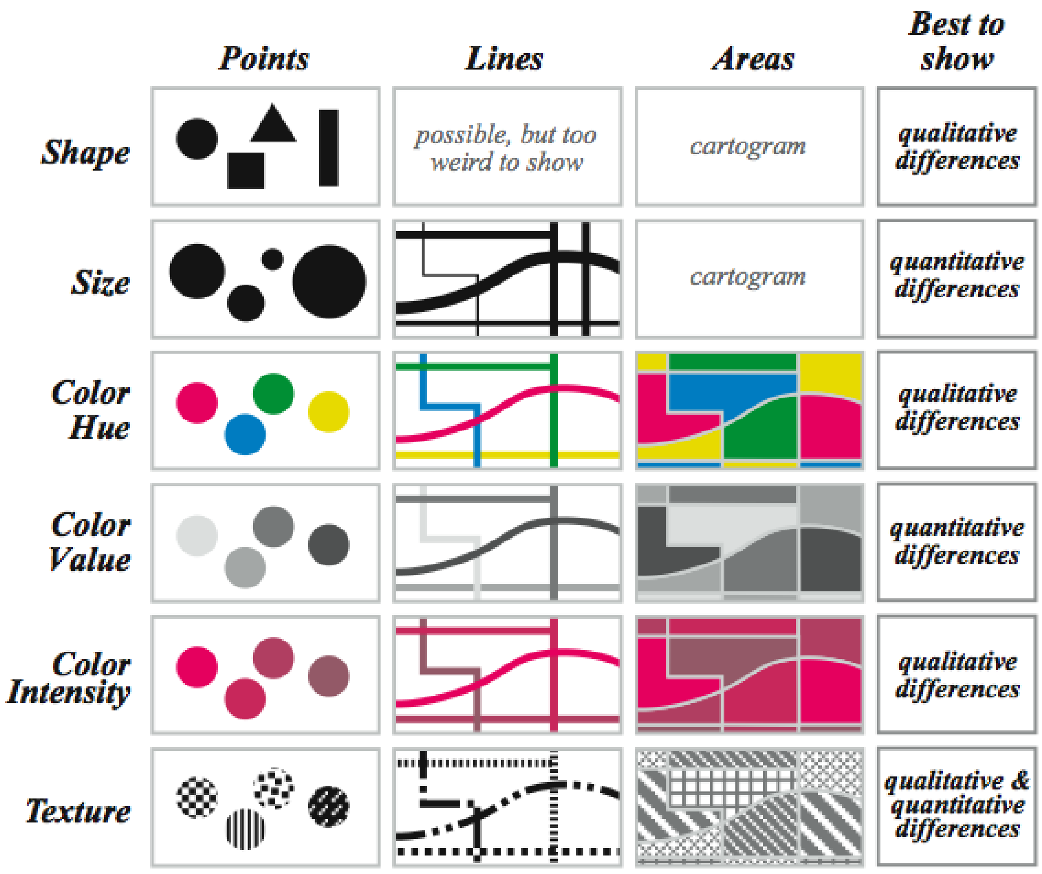


Figure 4: Retinal variables are a codification of how position, size, shape, color and texture are used to illustrate variations in the components of a visualization. The best to show column describes which types of information can be expressed in the corresponding visual encoding. This tabular form of Bertin's retinal variables is from Understanding Graphics [30] who reproduced it from *Making Maps: A Visual Guide to Map Design for GIS* [26]

The visual properties were first codified by Bertin[5], who also described the types of data that paired with the visual properties shown in figure 4. It is at this encoding level that Mackinlay formalized expressiveness as a homomorphic mapping which preserves some binary operator from one domain to another [29]. A functional dependency framework for evaluating these equivariances was proposed by Sugibuchi et al [42], and an algbraic basis for visualization design and evaluation was proposed by Kindlmann and Scheideggar[25]. Vickers et al. propose a category theory framework[49] that extends the equivariance checks to layout, but is focused strictly on the design layer like the other mathematical frameworks.

The visual encodings are composited into the point, line, and area graphical marks, as shown in figure 4. Marks can be generalized to glyphs, which are graphical objects that convey one or more attributes of the data entity mapped to it[32, 50] and minimally need to be differentiable from other visual elements [55]. As retinal variables and marks are the building blocks of most visualizations, they must be fully expressible in a framework for building visualization tools. By building visualization concepts into the core architecture, developers can incorporate assessments of the visualization, such as as quality metrics[6] or invariance [25] of visualizations with respect to graphical encoding choices.

2.4 Contribution

This work presents a mathematical model of the transformation from data to graphic representation and a proof of concept implementation. Specifically, this work contributes

1. a functional oriented visualization tool architecture
2. topology-preserving maps from data to graphic
3. monoidal action equivariant maps from component to visual variable
4. algebraic sum such that more complex visualizations can be built from simple ones
5. prototype built on Matplotlib's infrastructure

In contrast to mathematical models of visualization that aim to evaluate visualization design, we propose a topological framework for building tools to build visualizations. We defer judgement of expressivity and effectiveness to developers building domain specific tools, but provide them the framework to do so.