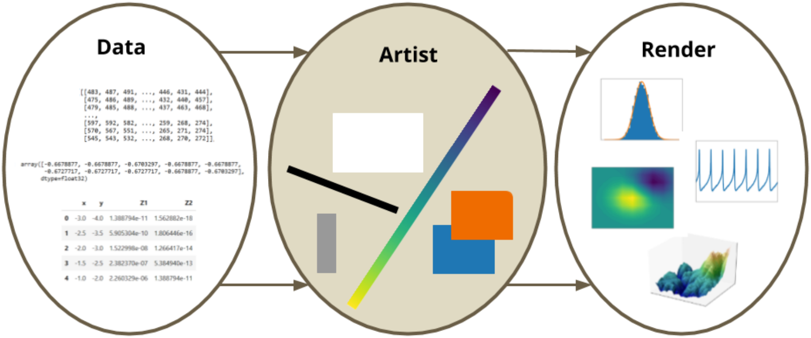
Topological Equivariant Artist Model

March 18, 2021

Hannah Aizenman
Advisor: Dr. Michael Grossberg
Committee: Dr. Robert Haralick, Dr. Lev Manovich, Dr. Huy Vo
External Member: Dr. Marcus Hanwell

Visualizations are structure preserving maps



The aim of this work is to rearchitecture Matplotlib to take advantage of developments in software design, data structures, and visualization to improve consistency, reusability, and discoverability, so domain specific tool developers can build structure preserving visualization tools.

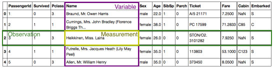
Visualization component constraints

equivariance properties of data and visual encoding match

continuity connectivity of data and visual encoding match

composibility structure preserved by individual components is preserved in combined components

Tools are tuned to date continuity [15]



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure: Based on fig 2.5 in Munzner's VAD[1]

1. Tableau[2–4]
2. ggplot[5]
3. Vega[6], Altair[7]

1. ImageJ[8], ImagePlot[9]
2. Napari[10]

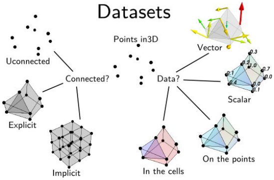


Figure: Data Representation, MayaVi 4.7.2 docs[11]

1. Gephi[12]
2. Graphviz[13]
3. Networkx[14]

Visualizations are tuned to data continuinity[23]

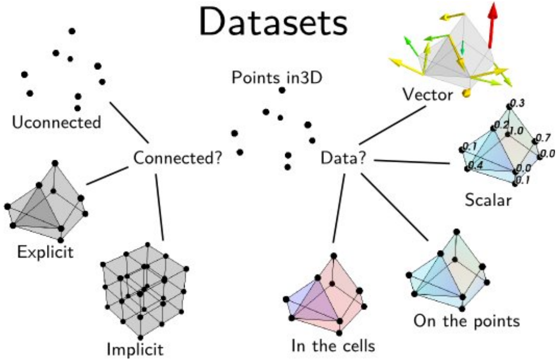


Figure: Data Representation, MayaVi 4.7.2 docs[11]

1. Matplotlib[16],
2. D3 [17]
3. VTK [18, 19], MayaVi[20], ParaView[21], Titan[22]

Structure is encoded in variables and continuity

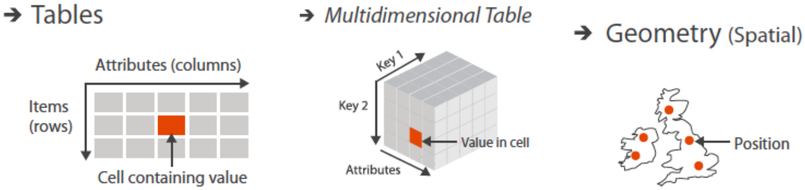


Figure: Image is figure 2.8 in Munzner's Visualization Analysis and Design[1]

binding
: metadata are structural *keys* with associated *values* (Munzner [1]))

continuity
: Fiber bundles can be a common data abstraction (Butler [24, 25])

variables
: Fibers can hold schema like encodings of variables (Spivak [26, 27])

Visualizations are (mostly) evaluated on equivariance

Expressiveness structure preserving mappings from data to graphic (Mackinlay [28])

Effectiveness design choices made in deference to perceptual saliency (Mackinlay [1, 29–31])

Naturalness easier to understand when properties match (Norman [32])

Graphical Integrity graphs show **only** the data (Tufte [33])

Models describe composition

language model APT, GoG: syntax, semantics, and grammar of graphics (Mackinlay, Wilkenson [28, 34, 35])

functional dependencies constrained maps between data and visual representation(Sugibuchi [36])

category theory the semiotics of visualization are commutative (Vickers [37])

algebraic process data (α) and viz (ω) transforms are symmetric (Kindlmann and Scheidegger [38])

- D data
- R representations
- V visualizations

$$\begin{tikzcd} D \arrow[r, "r_1"] \arrow[d, "\alpha"'] & R \arrow[r, "\nu"] & V \arrow[d, "\omega"] \\ D \arrow[r, "r_2"'] & R \arrow[r, "\nu"'] & V \end{tikzcd}$$

Contributions

Topological topology preserving relationship between data and graphic via continuous maps

Equivariant property preservation from data component to visual representation as equivariant maps that carry a homomorphism of monoid actions

Artist functional oriented visualization tool architecture built on the mathematical model to demonstrate the utility of the model

Model prototype of the architecture built on Matplotlib's infrastructure to demonstrate the feasibility of the model

Topological Equivariant Artist Model

The Artist $\mathscr{A}$ is a map from data $\mathscr{E}$ to graphic $\mathscr{H}$

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

that carries a homomorphism of monoid actions

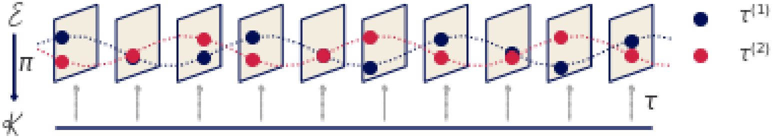
$$\varphi : M \to M' \tag{2}$$

such that artists are equivariant maps

$$\mathscr{A}(m \cdot r) = \varphi(m) \cdot \mathscr{A}(r) \tag{3}$$

with a deformation retraction from graphic to data space.

Data Bundle



A fiber bundle is a tuple (E, K, π, F) defined by the projection map π

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

where E is the total data space, F is the variable space, and K encodes the continuity.

Variables: Fiber

Given a space of all possible values $\mathbb{U}$

$$\begin{tikzcd} \mathbb{U}_\sigma \arrow[r] \arrow[d, "\pi_\sigma"'] & \mathbb{U} \arrow[d, "\pi"] \\ C \arrow[r, "\sigma"'] & \mathbf{DT} \end{tikzcd} \tag{5}$$

a fiber component is the restricted space $\mathbb{U}_{\sigma(c)}$.

$$F = \mathbb{U}_{\sigma(c)} = \mathbb{U}_T \tag{6}$$

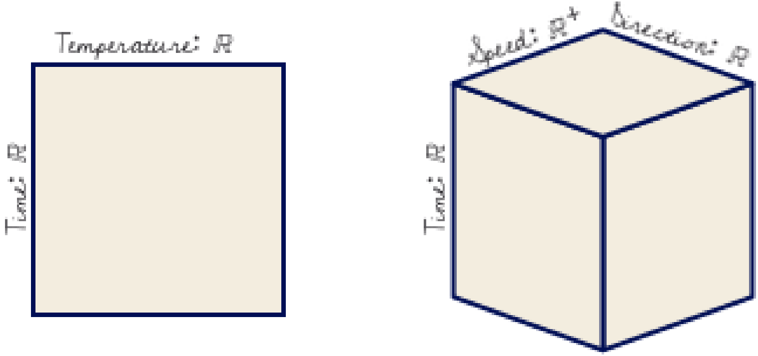
DT data types of the variables in the dataset

$\mathbb{U}$ disjoint union of all values of type $T \in \mathbf{DT}$

C variable names, $c \in C$

$\mathbb{U}_\sigma$ $\mathbb{U}$ restricted to the data type of a named variable

Variable types are dimensions of the fiber



Figure

plane $F = \mathbb{R} \times \mathbb{R}$, (time, temperature)

cube $\mathbb{R} \times \mathbb{R}^{+} \times \mathbb{R}$, (time, wind=(speed, direction))

Structure of Components: Monoid & Monoid Actions

A monoid M is a set with

associative binary operator $* : M \times M \to M$

identity element $e \in M$ such that $e * a = a * e = a$ for all $a \in M$.

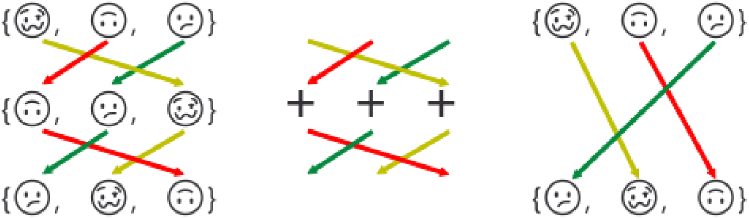
left monoid action

A set F with an action[39] $\bullet : M \times F \to F$ with the properties:

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

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

Monoid Actions: Permutation



Why monoids? partial orders

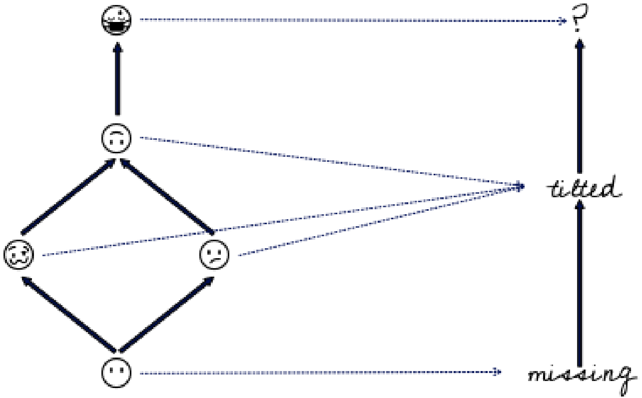
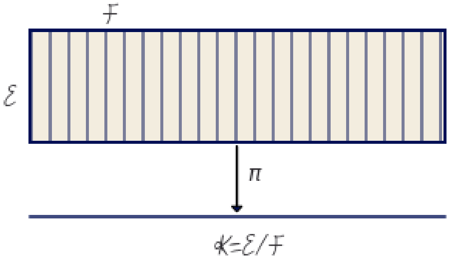


Figure: Inspired by definition 1.59 diagram in Spivak and Fong's An Invitation to Applied Category Theory [40]

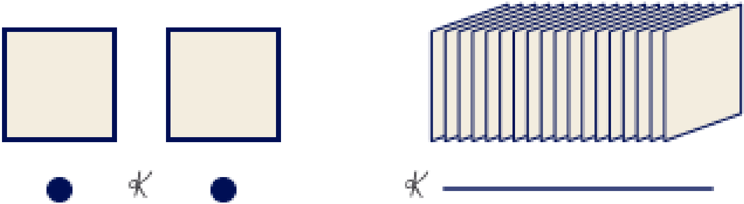
Data Continuity: Base space



where the total space can be decomposed into components

$$\pi : E_1 \oplus \ldots \oplus E_i \oplus \ldots \oplus E_n \rightarrow K \tag{7}$$

Data connectivity is encoded as the base space



Figure

points data is 0D discrete

line data is lies on the 1D continuous interval K

Values: Section

For any fiber bundle, there exists a map

$$\begin{tikzcd} F \arrow[r, hook] & E \arrow[d, "\pi"'] \\ & K \arrow[u, bend right, "\tau"'] \end{tikzcd} \tag{8}$$

s.t. $\pi(\tau(k)) = k$. Set of all global sections is denoted $\Gamma(E)$.

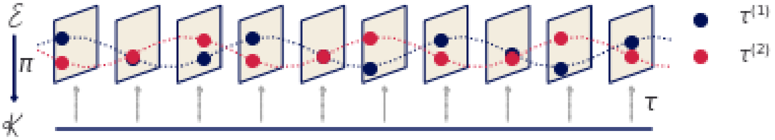
Record

Assuming a trivial fiber bundle $E = K \times F$, the section is

$$\tau(k) = (k, (g_{F_0}(k), \ldots, g_{F_n}(k))) \tag{9}$$

where $g : K \to F$ is the index function into the fiber.

Sample dataset



- F is $\mathbb{R} \times \mathbb{R}$
- K is interval $[0, 1]$
- $\tau^{(1)}$ is a *sin* function
- $\tau^{(2)}$ is a *cos* function
- $\tau^{(1)}, \tau^{(2)} \in \Gamma(E)$

Sheafs

Restriction maps of a sheaf describe how local $\iota^*\tau$ can be glued into larger sections [41, 42].

$$\begin{tikzcd} \iota^* E \arrow[r, hook, "\iota^*"] \arrow[d, "\pi"'] & E \arrow[d, "\pi"'] \\ U \arrow[r, hook, "\iota"] \arrow[u, bend right, "\iota^*\tau"'] & K \arrow[u, bend right, "\tau"'] \end{tikzcd} \tag{10}$$

The inclusion map $\iota : U \to K$ pulls E over U such that the pulled back $\iota^*\tau$ only contains records over $U \subset K$.

Graphic Bundle

The graphics bundle is a tuple (H, S, π, D) defined by the projection map π

$$\begin{tikzcd} D \arrow[r, hook] & H \arrow[d, "\pi"'] \\ & S \arrow[u, bend right, "\rho"'] \end{tikzcd} \tag{11}$$

where ρ is the fully encoded graphic.

Example: 2D opaque image

The target display is $D = \mathbb{R}^5$ with elements

$$(x, y, r, g, b) \in D$$

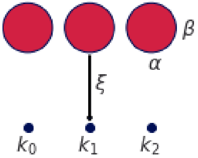
returned by ρ such that a graphic has color and 2D position.

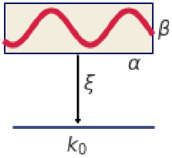
Graphic Continuity

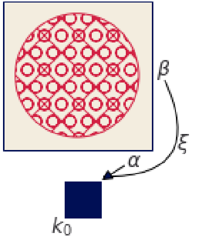
The surjective map $\xi : S \to K$

$$\begin{tikzcd} E \arrow[d, "\pi"'] & H \arrow[d, "\pi"'] \\ K & S \arrow[l, "\xi"'] \end{tikzcd} \tag{12}$$

goes from region $s \in S_k$ to its associated point k in data space.







Topological Equivariant Artist Model

The topological artist A is a monoid equivariant sheaf map

$$\begin{tikzcd} E' \arrow[r, "\nu"] \arrow[dr, "\pi"'] & V \arrow[d, "\pi"] & \xi^* V \arrow[l, "\xi^*"'] \arrow[r, "Q"] \arrow[d, "\xi^*\pi"'] & H \arrow[dl, "\pi"] \\ & K & S \arrow[l, "\xi"'] & \end{tikzcd} \tag{13}$$

where the artist $A : \mathcal{O}(E) \to \mathcal{O}(H)$ takes as input $E' = \mathcal{J}^2(E)$.

Visual Bundle

The visual bundle is a tuple (V, K, π, P) defined by the projection map π

$$\begin{tikzcd} P \arrow[r, hook] & V \arrow[d, "\pi"'] \\ & K \arrow[u, bend right, "\mu"'] \end{tikzcd} \tag{14}$$

where μ is the visual variable encoding[43] of the data section τ.

Example: position and color

Given an artist with parameters $\{xpos, ypos, color\}$, a sample visual section μ could be $\{.5, .5, (255, 20, 147)\}$

Visual Channel Encoders

We define the visual transformers ν on components of the data bundle τ_i

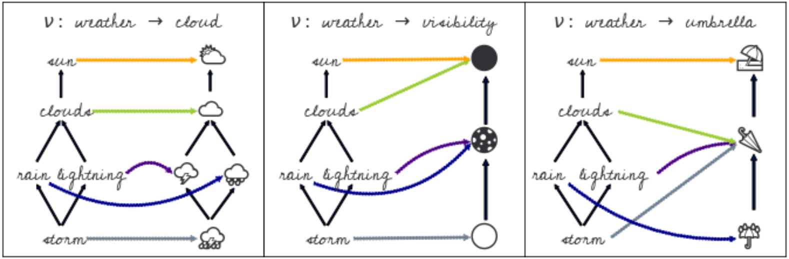
$$\{\nu_0, \ldots, \nu_n\} : \{\tau_0, \ldots, \tau_n\} \mapsto \{\mu_0, \ldots, \mu_n\} \tag{15}$$

as the set of equivariant maps with the constraint

$$\nu_i(m_r(E_i)) = \varphi(m_r)(\nu_i(E_i)) \tag{16}$$

where $\varphi : M \to M'$ carries a homomorphism of monoid actions.

Example: Partial Order Equivariance



Measurement Scale Groups

nominal	permutation	if $r_1 \neq r_2$ then $\nu(r_1) \neq \nu(r_2)$
ordinal	monotonic	if $r_1 \leqslant r_2$ then $\nu(r_1) \leqslant \nu(r_2)$
interval	translation	$\nu(x + c) = \nu(x) + c$
ratio	scaling	$\nu(xc) = \nu(x) * c$

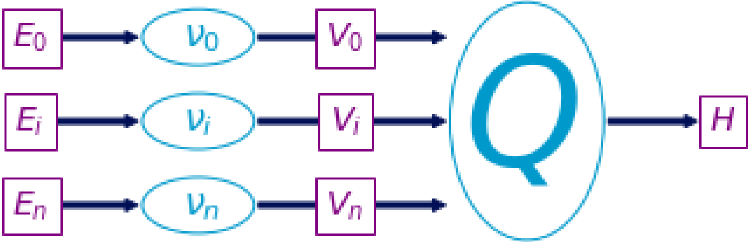
Invalid ν

Given $\nu_i(x) = .5$ and $t(x) = x + 2$,

$$\nu(t(r+2)) \stackrel{?}{=} \nu(r) + 2$$

$$.5 \neq .5 + 2$$

Visualization Assembly Function



Glyph

The glyph is the graphic generated by $Q(S_j)$ where the path connected components $J \subset K$ are defined

$$J = \{j \in K \text{ s. t. } \exists \gamma \text{ s.t. } \gamma(0) = k \text{ and } \gamma(1) = j\} \qquad (17)$$

such that the path γ from k to j is a continuous function from the interval [0,1] and S_j is the region

$$H \underset{\rho(S_j)}{\overset{}{\leftrightarrows}} S_j \underset{\xi^{-1}(J)}{\overset{\xi(s)}{\leftrightarrows}} J_k \qquad (18)$$

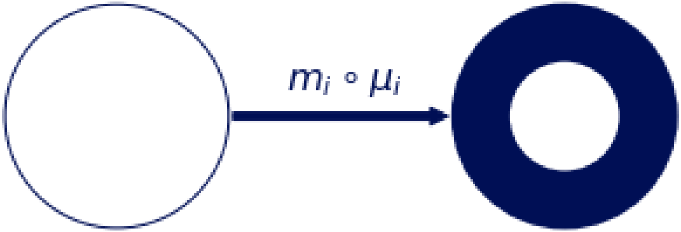
such that the glyph is differentiable, in keeping with Ziemkiewicz and Kosara's description of a glyph[44].

Visualization Equivariance

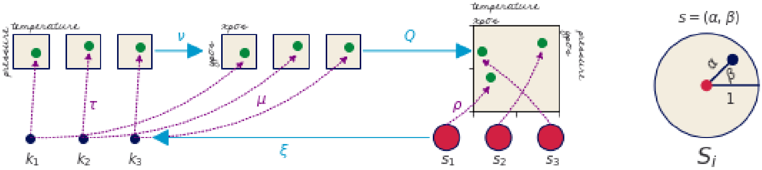
If Q is applied to μ, μ' that generate the same ρ

$$Q(\mu) = Q(\mu') \implies Q(m \circ \mu) = Q(m \circ \mu') \tag{19}$$

then the output of both sections acted on by the same monoid m must be the same.



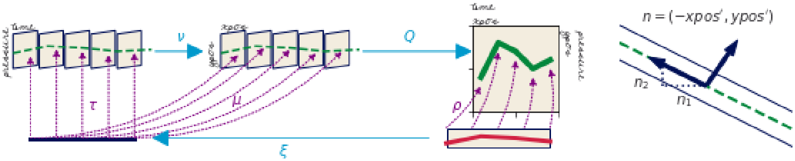
Scatter: $Q(xpos, ypos)(\alpha, \beta)$



$$x = size * \alpha \cos(\beta) + xpos$$

$$y = size * \alpha \sin(\beta) + ypos$$

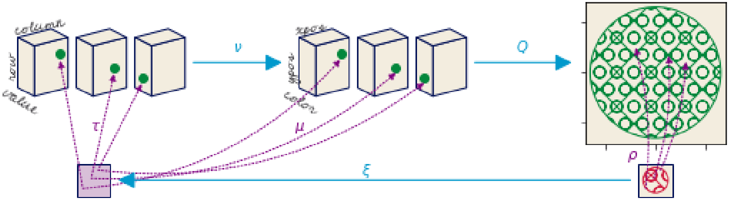
Line: $Q(xpos, \hat{n}_1, ypos, \hat{n}_2)(\alpha, \beta)$



$$|n| = \sqrt{{n_1}^2 + {n_2}^2},\ \ \hat{n}_1 = \frac{n_1}{|n|},\ \ \hat{n}_2 = \frac{n_2}{|n|}$$

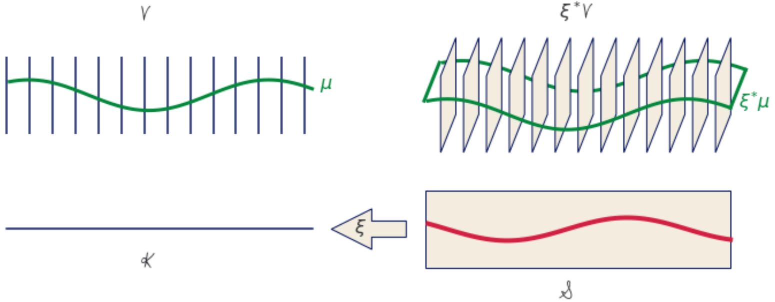
$$x = xpos(\xi(\alpha)) + width * \beta\,\hat{n}_1(\xi(\alpha))$$
$$y = ypos(\xi(\alpha)) + width * \beta\,\hat{n}_2(\xi(\alpha))$$

Image $Q(xpos, ypos, color)$



$$R = R(\xi(\alpha, \beta))$$
$$G = G(\xi(\alpha, \beta))$$
$$B = B(\xi(\alpha, \beta))$$

Assembly Function Factory



$$\hat{Q}(\mu(k))(s) \coloneqq Q((\xi^*\mu)(s)) \tag{20}$$

such s can be factored out when $\xi^{-1}(k) = s$

Composition of artists

Given the family of artists $(E_i : i \in I)$ on the same image

$$+ \coloneqq \bigsqcup_{i \in I} E_i \tag{21}$$

the $+$ operator defines a simple composition of artists.
When artists share a base space

$$K_2 \hookrightarrow K_1 \tag{22}$$

a composition operator can be defined such that the the artists can be considered to be acting on different components of the same section.

Equivalance class of artists

An approximation of the equivalence class of artists A'

$$A \in A' : A_1 \equiv A_2 \tag{23}$$

roughly treats two artists as equivalent if they

- act on the same visual bundle V
- have the same assembly function Q
- have the same continuity map ξ

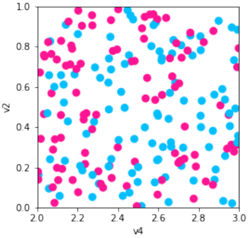
Artist

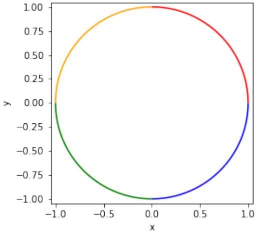
```python
class ArtistClass(matplotlib.artist.Artist):
    def __init__(self, data, transforms, *args, **kwargs):
        # properties that are specific to the graphic
        self.data = data
        self.transforms = transforms
        super().__init__(*args, **kwargs)

    def assemble(self, **args):
        # set the properties of the graphic

    def draw(self, renderer):
        # returns K, indexed on fiber then key
        view = self.data.view(self.axes)
        # visual channel encoding applied fiberwise
        visual = {p: t['encoder'](view[t['name']])
                    for p, t in self.transforms.items()}
        self.assemble(**visual)
        # pass configurations off to the renderer
        super().draw(renderer)
```

Artists: Scatter & Line






```
fig, ax = plt.subplots()
artist = Point(data, transforms)
ax.add_artist(artist)
```

```
fig, ax = plt.subplots()
artist = Line(data, transforms)
ax.add_artist(artist)
```

Artists: Scatter & Line

```
class Point(mcollections.Collection):
    def assemble(self, x, y, s, facecolors='C0' ):
        # construct geometries of the circle glyphs in visual coordinates
        self._paths = [mpath.Path.circle(center=(xi,yi), radius=si)
                       for (xi, yi, si) in zip(x, y, s)]
        # set attributes of glyphs, these are vectorized
        # circles and facecolors are lists of the same size
        self.set_facecolors(facecolors)
```

```
class Line(mcollections.LineCollection):
    def assemble(self, x, y, color='C0'):
        #assemble line marks as set of segments
        segments = [np.vstack((vx, vy)).T for vx, vy in zip(x, y)]
        self.set_segments(segments)
        self.set_color(color)
```

Visual Transformations

```
cmap =  color.Categorical({'true':'deeppink', 'false':'deepskyblue'})
transforms = {'x': {'name': 'v4', 'encoder': lambda x: x},
                'y': {'name': 'v2', 'encoder': lambda x: x},
                'facecolors': {'name':'v3', 'encoder': cmap},
                's':{'name': None ,
                    'encoder': lambda _: itertools.repeat(.02)}}
```

- `lambda x: x` is identity ν
- `{'name':None}` map into P without corresponding τ
- `color.Categorical` is custom ν

Custom Complex ν

```
class Categorical:
    def __init__(self, mapping):
        # check that the conversion is to valid colors
        assert(mcolors.is_color_like(color) for color in mapping.values())
        self._mapping = mapping

    def __call__(self, value):
        # convert value to a color
        return [mcolors.to_rgba(self._mapping[v]) for v in values]
```

That we can test for action equivariance

```
def test_nominal(values, encoder):
    m1 = list(zip(values, encoder(values)))
    random.shuffle(values)
    m2 = list(zip(values, encoder(values)))
    assert sorted(m1) == sorted(m2)
```

Fiber Bundle

```
@dataclass
class FiberBundle:
"""
Attributes
----------
K: {'tables': []}
F: {variable name: type}
"""
    K: dict
    F: dict
```

Discrete Connectivity

```
class VertexSimplex: #maybe change name to something else
    """Fiberbundle is consistent across all sections
    """
    FB = FiberBundle({'tables': ['vertex']},
            {'v1': float, 'v2': str, 'v3': float})

    def __init__(self, sid = 45, size=1000, max_key=10**10):
        # create random list of keys
    def tau(self, k):
        # e1 is sampled from F1, e2 from F2, etc...
        return (k, (e1, e2, e3, e4))
```

1D Continuous Connectivity

```
class EdgeSimplex:
    FB = FiberBundle({'tables': ['vertex','edge']},
                     {'x' : float, 'y':  float,
                     'color':mtypes.Color()}})
    def __init__(self, num_edges=4, num_samples=1000):
        self.keys = range(num_edge) #edge id
        self.distances = np.linspace(0,1, num_samples)
        # half generlized representation of arcs on a circle
        self.angle_samples = np.linspace(0, 2*np.pi, len(self.keys)+1)
    @staticmethod
    def _color(edge):
        return ['red','orange', 'green','blue'][edge%4]
    @staticmethod
    def _xy(edge, distances, start=0, end=2*np.pi):
        # start and end are parameterizations b/c really there is
        angles = (distances *(end-start)) + start
        return np.cos(angles), np.sin(angles)
    def tau(self, k): #will fix location on page on revision
        x, y = self._xy(k, self.distances,
                        self.angle_samples[k], self.angle_samples[k+1])
        color = self._color(k)
        return (k, (x, y, color))
```

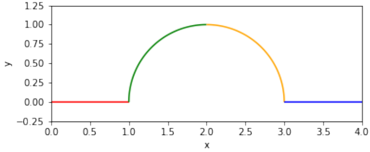
View

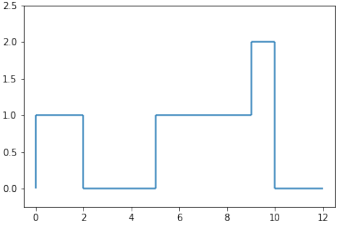
```python
def view(self, axes):
    table = defaultdict(list)
    for k in self.keys:
        table['index'].append(k)
        for (name, value) in zip(self.FB.fiber.keys(), self.tau(k)[1]):
            table[name].append(value)
    return table
```

`VertexSimplex` (name, value), value is scaler

`EdgeSimplex` (name, value), value is [x0, ..., xn]

Same Artist, Different Data Configurations






```
simplex.GraphLine(FB, edge_table, vertex_table,
                              connect=True)
simplex.GraphLine(FB, edge_table, vertex_table,
                              num_samples=2, connect=False)
```

Summary

- structure preserving maps from data to visual representation:
  - data and graphics have equivalent continuity
  - properties are equivariant under monoid actions
- fiber bundles with a schema are structure rich abstractions of
  - topologically complex heterogenous data
  - target display spaces
- model can be iteratively integrated into existing Matplotlib architecture

Proposed Dissertation

- expansion of the mathematical framework to include worked out simple and complex addition
- formalization of definition of equivalance class A'
- implementation of artist with explicit ξ
- specification of interactive visualization
- mathematical formulation of a graphic with axes labeling
- implementation of new prototype artists that do not inherit from Matplotlib artists
- provisional mathematics and implementation of user level composite artists
- proof of concept domain specific user facing library

References I

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

[2] C. Stolte, D. Tang, and P. Hanrahan. "Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (Jan. 2002), pp. 52–65.

[3] P. Hanrahan. "VizQL: A Language for Query, Analysis and Visualization". In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 721.

References II

[4] J. Mackinlay, P. Hanrahan, and C. Stolte. "Show Me: Automatic Presentation for Visual Analysis". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1137–1144.

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

[6] A. Satyanarayan, K. Wongsuphasawat, and J. 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.

[7] J. VanderPlas et al. "Altair: Interactive Statistical Visualizations for Python". en. In: *Journal of Open Source Software* 3.32 (Dec. 2018), p. 1057.

References III

[8] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. "NIH Image to ImageJ: 25 Years of Image Analysis". In: *Nature Methods* 9.7 (July 2012), pp. 671–675.

[9] S. Studies. *Culturevis/Imageplot*. Jan. 2021.

[10] N. Sofroniew et al. *Napari/Napari: 0.4.5rc1*. Zenodo. Feb. 2021.

[11] *Data Representation in Mayavi — Mayavi 4.7.2 Documentation*. https://docs.enthought.com/mayavi/mayavi/data.html.

[12] M. Bastian, S. Heymann, and M. Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". en. In: *Proceedings of the International AAAI Conference on Web and Social Media* 3.1 (Mar. 2009).

References IV

[13] J. Ellson et al. "Graphviz— Open Source Graph Drawing Tools". In: *Graph Drawing*. Ed. by P. Mutzel, M. Jünger, and S. Leipert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 483–484.

[14] A. A. Hagberg, D. A. Schult, and P. J. Swart. "Exploring Network Structure, Dynamics, and Function Using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, 2008, pp. 11–15.

[15] J. Heer and M. Agrawala. "Software Design Patterns for Information Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 853–860.

[16] J. D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3 (May 2007), pp. 90–95.

[17] M. Bostock, V. Ogievetsky, and J. Heer. "D^3 Data-Driven Documents". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (Dec. 2011), pp. 2301–2309.

[18] M. 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.

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

[20] P. Ramachandran and G. Varoquaux. "Mayavi: 3D Visualization of Scientific Data". In: *Computing in Science Engineering* 13.2 (Mar. 2011), pp. 40–51.

References VI

[21] J. Ahrens, B. Geveci, and C. Law. "Paraview: An End-User Tool for Large Data Visualization". In: *The visualization handbook* 717.8 (2005).

[22] Brian Wylie and Jeffrey Baumes. "A Unified Toolkit for Information and Scientific Visualization". In: *Proc.SPIE*. Vol. 7243. Jan. 2009.

[23] M. Tory and T. Moller. "Rethinking Visualization: A High-Level Taxonomy". In: *IEEE Symposium on Information Visualization*. 2004, pp. 151–158.

[24] D. M. Butler and S. Bryson. "Vector-Bundle Classes Form Powerful Tool for Scientific Visualization". en. In: *Computers in Physics* 6.6 (1992), p. 576.

[25] 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.

References VII

[26] D. I. Spivak. *Databases Are Categories*. en. Slides. June 2010.

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

[28] J. Mackinlay. "Automating the Design of Graphical Presentations of Relational Information". In: *ACM Transactions on Graphics* 5.2 (Apr. 1986), pp. 110–141.

[29] W. S. Cleveland. "Research in Statistical Graphics". In: *Journal of the American Statistical Association* 82.398 (June 1987), p. 419.

[30] W. S. Cleveland and R. McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods". In: *Journal of the American Statistical Association* 79.387 (Sept. 1984), pp. 531–554.

References VIII

[31] J. M. Chambers et al. *Graphical Methods for Data Analysis*. Vol. 5. Wadsworth Belmont, CA, 1983.

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

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

[34] J. Mackinlay. "Automatic Design of Graphical Presentations". English. PhD Thesis. Stanford, 1987.

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

[36] T. Sugibuchi, N. Spyratos, and E. Siminenko. "A Framework to Analyze Information Visualization Based on the Functional Data Model". In: *2009 13th International Conference Information Visualisation*. 2009, pp. 18–24.

[37] P. Vickers, J. Faith, and N. Rossiter. "Understanding Visualization: A Formal Approach Using Category Theory and Semiotics". In: *IEEE Transactions on Visualization and Computer Graphics* 19.6 (June 2013), pp. 1048–1061.

[38] G. Kindlmann and C. Scheidegger. "An Algebraic Process for Visualization Design". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 2181–2190.

[39] nLab authors. "Action". In: (Mar. 2021).

[40] B. Fong and D. I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. en. First. Cambridge University Press, July 2019.

References X

[41] R. W. Ghrist. *Elementary Applied Topology*. Vol. 1. Createspace Seattle, 2014.

[42] R. Ghrist. "Homological Algebra and Data". In: *Math. Data* 25 (2018), p. 273.

[43] J. Bertin. "II. The Properties of the Graphic System". English. In: *Semiology of Graphics*. Redlands, Calif.: ESRI Press, 2011.

[44] C. Ziemkiewicz and R. Kosara. "Embedding Information Visualization within Visual Representation". In: *Advances in Information and Intelligent Systems*. Ed. by Z. W. Ras and W. Ribarsky. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 307–326.

Rendering: Define a Pixel

Given a pixel

$$p = \left[y_{top}, y_{bottom}, x_{right}, x_{left}\right] \tag{24}$$

the inverse map of the bounding box

$$S_p = \rho_{xy}{}^{-1}(p) \tag{25}$$

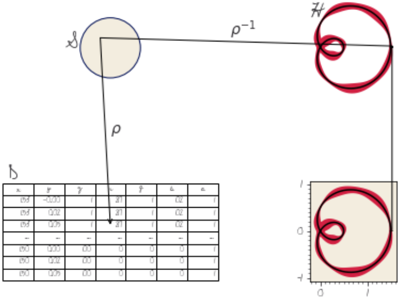
is a region $S_p \subset S$ such that

$$r_p = \iint_{S_p} \rho_r(s) ds^2 \tag{26}$$

$$g_p = \iint_{S_p} \rho_g(s) ds^2 \tag{27}$$

$$b_p = \iint_{S_p} \rho_b(s) ds^2 \tag{28}$$

yields the color of the pixel.



P Components

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

GraphLine Data Model

```
class GraphLine:
    def __init__(self, FB, edge_table, vertex_table, num_samples=1000,
                        connect=False):
        #set args as attributes and generate distance
        if connect: # test connectivity if edges are continuous
            assert edge_table.keys() == self.FB.F.keys()
            assert is_continuous(vertex_table)

    def tau(self, k):
        # evaluates functions defined in edge table
        return(k, (self.edges[c][k](self.distances)
                        for c in self.FB.F.keys()))

    def view(self, axes):
        # walk the edge_vertex table to return the edge function
        table = defaultdict(list)
        for (i, (start, end)) in sorted(zip(self.ids, self.vertices),
                                        key=lambda v:v[1][0]):
            table['index'].append(i)
            # same as view for line, returns nested list
            for (name, value) in zip(self.FB.F.keys(), self.tau(i)[1]):
                table[name].append(value)
        return table
```