Comparative Visualizations through Parameterization and Variability

Karl Smeltzer
Oregon State University
karl@lifetime.oregonstate.edu

Martin Erwig
Oregon State University
erwig@oregonstate.edu

***Abstract*—Comparative visualizations and the comparison tasks they support constitute a crucial part of visual data analysis on complex data sets. Existing approaches are ad hoc and often require significant effort to produce comparative visualizations, which is impractical especially in cases where visualizations have to be amended in response to changes in the underlying data.**

We show that the combination of parameterized visualizations and variations yields an effective model for comparative visualizations. Our approach supports data exploration and automatic visualization updates when the underlying data changes. We provide a prototype implementation and demonstrate that our approach covers most of existing comparative visualizations.

I. Introduction

To make sense of the fast-growing amounts of data, information visualization is getting more and more important. The rate of data collection in general is growing exponentially, driven by the rise of technologies such as autonomous vehicles and smart devices [1]. In turn, this continues to drive the development of new approaches and techniques in data visualization to explore and explain the data.

Comparative visualization is one such approach that focuses specifically on comparison tasks when analyzing data. Comparisons tasks feature prominently in all kinds of visualization history mechanisms [2], uncertainty visualization [3], software visualization [4], and many more areas.

The widely adopted approach of using small multiples [5] (roughly, composing the variant visualizations into a grid containing all possibilities) provides only a partial solution. Two immediate problems of this approach are how to organize the charts into a grid and how to ensure they are simple enough to be read at a small size. More seriously, a grid layout is inherently only scalable in two dimensions. As the number of orthogonal parameters in the data grows we need exponentially many charts to keep up.

Another complication arises from the difficulty of context-dependent comparison. For example, suppose we are tasked with an analysis of profits per quarter for a business. During that process we might need to undertake some semantic zooming subtasks such as comparing only fourth quarter profits across years to see the impact of holiday bonuses, or perhaps exploring the data only for specific geographic regions. Such tasks can of course always be performed manually by returning to the data, subsetting or manipulating it, and then re-creating appropriate visualizations. However, it is highly inefficient to essentially having to start from scratch with each iteration.

978-1-5386-4235-1/18/$31.00

In this work we propose a model for creating, transforming, and comparing visualizations based on the notion of variation that helps to systematize how data scientists can approach these challenges. We begin by revisiting a model for constructing traditional, non-variational visualizations in Section II. In Section III we review a model of variation as well as its application to represent variational pictures. Built on these two components, we introduce variational visualizations in Section IV with numerous examples. In Section V we evaluate how our model of variational visualizations supports comparative visualization. Section VI discusses related work and Section VII presents some conclusions. This work makes the following specific contributions.

- An *expressive model* of variational visualizations.
- A *prototype* implementation of the model, used to generate all of the example figures in this paper.
- An *evaluation* of the model's suitability for comparative visualization tasks.

II. A Model for Data Visualization

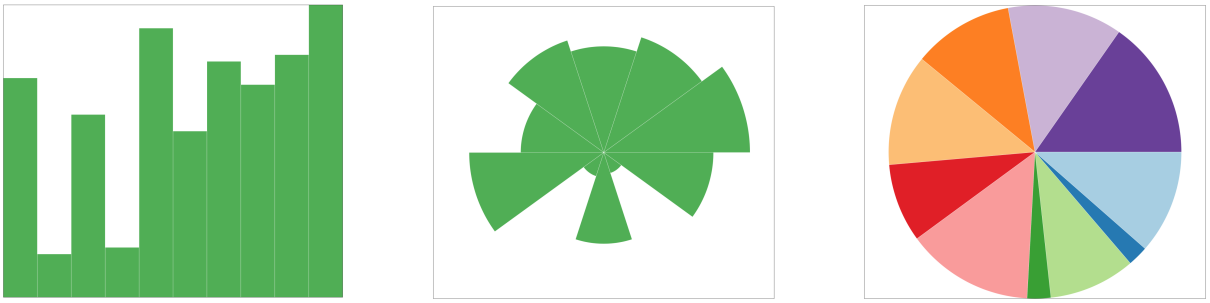
We build on the visualization approach presented in [6]. Here we focus on a small subset of visualization types with the goal of systematizing how we construct and compose them.

A visualization is essentially a composition of *marks*. Marks encode primitive shapes implicitly through visual parameter mappings. Based on Bertin's visual variable [7], visual parameters are any rendered aspects of a mark that can be bound to a data value, such as color, size, location, orientation, etc. Marks also contain labeling information.

$$\mathit{Mark} \quad = \quad \mathit{VisualParameter}^* \times \mathit{Label}$$

A visualization is essentially given by a composition of marks, a transformation between coordinate systems or an overlay of visualizations. For simplicity we employ a simple prefix notation (allowing binary operations to be written infix).

$$\begin{array}{lcl} \mathit{Visualization} & ::= & \mathit{Mark} \\ & | & \textbf{NextTo}\ \mathit{Visualization}^* \\ & | & \textbf{Above}\ \mathit{Visualization}^* \\ & | & \textbf{Cartesian}\ \mathit{Visualization} \\ & | & \textbf{Polar}\ \mathit{Visualization} \\ & | & \textbf{Overlay}\ \mathit{Visualization}^* \\ & | & \ldots \end{array}$$



(a) (b) (c)

Fig. 1: A series of non-variational and non-comparative visualizations constructed in our prototype using composition and transformation. Both (b) and (c) are built from the visualization in (a) as a starting point.

In a Cartesian coordinate system the **NextTo** and **Above** constructs divide space horizontally and vertically, respectively. In a polar coordinate system they divide the space by angle and radius, respectively.

For example, to construct a bar chart we need a sequence of marks in which the height visual parameter of the primitive rectangles is bound to a data value. We can also set the color parameter of the marks and attach labels.

$$m_1 = ([height \mapsto 6.6, color \mapsto green, \ldots], \text{“6.6”})$$
$$\vdots$$
$$m_{10} = ([height \mapsto 8.1, color \mapsto green, \ldots], \text{“8.1”})$$

We can then compose these horizontally to generate a simple chart shown in Figure 1a.[1]

$$bars = \textbf{NextTo}\ [m_1, \ldots, m_n]$$

Our model provides many shortcuts to avoid the tedious construction of the individual marks. For example:

$$bars = barchart\ [6.6, \ldots, 8.1]\ \text{`colorAll`}\ green$$

Building visualizations from composable parts makes it easy to transform them. For example, suppose we would like to see whether a Coxcomb chart (essentially a radial area chart, where data is bound to the radius of equal-angle wedges) might be more appropriate than a bar chart. In a typical visualization tool this would require either starting over or perhaps copying and modifying the code responsible for generating it.

Instead, our model allows visualizations to be transformed directly, which avoids the need for managing multiple artifacts or code snippets. In this particular case, a Coxcomb chart is simply a bar chart reinterpreted in a polar coordinate system. The marks are still composed next to one another (because in a polar environment, horizontal composition corresponds to the angle), and the data bound to the height does not need to change (since vertical height corresponds to the radius distance in the polar system). The rendered output of this transformation is shown in Figure 1b.

$$cox = \textbf{Polar}\ bars$$

[1]All visualizations in this paper have been created with a prototype implementation of a visualization DSL that can be found at: https://github.com/karljs/vis.

If we find the Coxcomb chart too difficult to read, we can turn it into a pie chart instead. In a typical visualization tool we again would likely need to start over. But because a pie chart corresponds closely to a Coxcomb chart, we can produce one easily using another transformations. We still want to compose our marks next to one another in a polar environment, but we want to change the data to map to the angle (or width) instead of the radius (or height). Since this requires modifying the visual parameters, we need more than just another composition operator. For this purpose we provide a series of functions that modify visualizations, such as *reorient*, which flips the width and height parameters of all the marks in a visualization for us.

$$reorient : Visualization \rightarrow Visualization$$

Applying the *reorient* operation gives us the pie chart we were expecting, but there is one more thing we can do. Pie charts are often more effective when the individual wedges are colored distinctly rather than with a single color, to provide some visual separation. We could choose new colors manually, but our system also defines some color schemes that can be applied automatically. We can use the *color* operation to recolor the chart, which takes a sequence of colors as a parameter. The simplest way to use a nice default qualitative color scheme we can just pass it the built-in *defaultColors*.

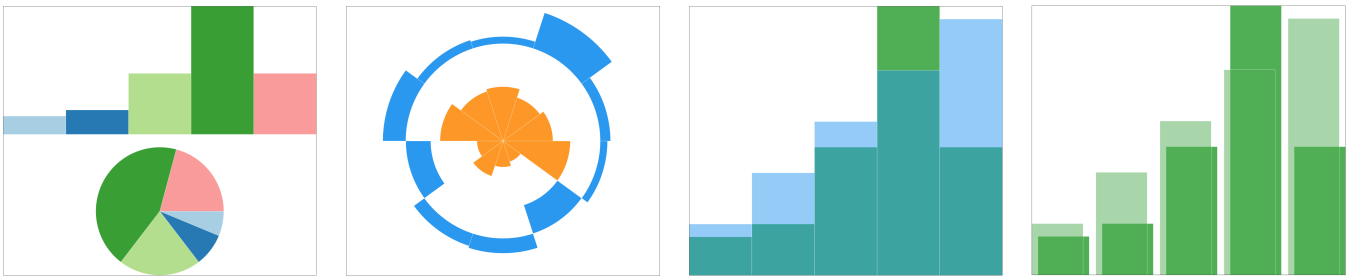
$$pie = reorient\ cox\ \text{`color`}\ defaultColors$$

This produces the rendered result shown in Figure 1c. There are many more useful transformations that are possible; more detail is provided in [6].

One important feature of the presented visualization model is its ability to define and apply functions, which allows visualization to be parameterized. Parameterization provides a dynamic form of variationalization.

A. Comparative Visualizations Without Variability

It is not always necessary to use variation to compare two visualization designs. Using the operation of our visualization model, we can also already generate a limited form of comparative visualizations using operators such as **NextTo** and **Above**. For example, we could compare a barchart *bar*



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

Fig. 2: Examples of comparative visualizations constructed in our prototype which do not make use of variability.

with its equivalent pie chart by explicitly placing one above the other, as shown in Figure 2a.

$$\textbf{Above}\ [bar,\ reorient\ (\textbf{Polar}\ bar)]$$

Finally, when using polar charts, such as Coxcomb charts, we can align wedges in concentric rings, when appropriate. This makes comparing two polar charts more reasonable than putting them beside one another.

$$\textbf{Polar}\ (\textbf{Above}\ [cox_1,\ cox_2])$$

Such a chart can be seen in Figure 2b.

Instead of showing visualizations side-by-side, another possibility for comparison tasks is to show them overlaid. This can be achieved in the same way as spatial composition by using **Overlay**. However, to avoid the occlusion of parts of visualizations, we can employ transparency to ensure lower layers are always visible. An example is shown in Figure 2c.

Another approach is to partially offset the marks comprising the lower layer visualizations in order to prevent them from being totally obscured. In the following example, we make use of both approaches simultaneously. The *alpha* command configured the level of transparency for a visualization. Setting RGBA colors directly is also supported. The spacing is slightly more complicated. The relative spacing model is detailed in [6], but the important aspect to know here is that when space is applied it is always sized relative to a particular visualization or element. For example, if spacing is applied between the bars of a bar chart, it is specified as a ratio to the width of the bars. If spacing is inserted between entire visualizations, it is sized as a ratio of the width of the entire visualizations. In the following example, we apply spacing both between bars and on the edges of entire visualizations in order to provide enough space for all bars to show.

The *space* function places empty space between the elements of the visualization to which it is applied. In this case, we apply it to a composition of bars with the argument 0.25, which means that it will create a space equivalent to one-quarter the width of the bars between each pair of bars. The *rightSpace* and *leftSpace* functions behave slightly differently. Those essentially compose empty space onto one side of an entire visualization. For example, when we apply *rightSpace* with the argument 0.02, it produces space equal to 2 percent of the visualization's width and composes it on the right. The internal spacing is unaffected.[2]

$$\textbf{Overlay}\ [bar_1\ \text{`}alpha\text{`}\ 0.5\ \text{`}space\text{`}\ 0.25\ \text{`}rightSpace\text{`}\ 0.02,\ bar_1\ \text{`}space\text{`}\ 0.25\ \text{`}leftSpace\text{`}\ 0.02]$$

Since our visualization model allows the definition of functions, we can identify patterns such as this and capture them in function definitions. Applying this visualization pattern to all bars leads to the result shown in Figure 2d.

III. Representing Variation

The choice calculus [8] is a formal model of variation built on the core concept of *choices*. Choices attach names, or *dimensions*, to a list of *alternatives*. For example, we can write a choice in dimension A between two variant numbers as $A\langle 1, 2\rangle$. Choices can also be nested, as in $A\langle B\langle 1, 2\rangle, 3\rangle$. In this paper we limit ourselves to binary choices for simplicity, since it is always possible to represent choices with more alternatives using a sequence of nested choices.

Each binary dimension D also leads to two *selectors*, $D.l$ and $D.r$. Selectors indicate particular branches in that dimensions and can be used for *selection*, which reduces or eliminates variability. Selection is defined as follows where s ranges over selectors, vx and vy range over variational values, and x ranges over plain values.

$$\lfloor D\langle vx, vy\rangle \rfloor_s = \begin{cases} \lfloor vx \rfloor_s & \text{if } s = D.l \\ \lfloor vy \rfloor_s & \text{if } s = D.r \\ D\langle \lfloor vx \rfloor_s, \lfloor vy \rfloor_s \rangle & \text{otherwise} \end{cases}$$

$$\lfloor x \rfloor_s = x$$

For example, $\lfloor A\langle B\langle 1, 2\rangle, 3\rangle \rfloor_{B.r} = A\langle 2, 3\rangle$. When multiple dimensions share choices they are *synchronized*, meaning that performing a selection for one automatically performs the same selection for all choices in that dimension.

Sets of selectors are called *decisions* and can be used to eliminate variation with repeated selection. For a decision $\delta = \{s_1, s_2, \ldots, s_n\}$, we have $\lfloor vx \rfloor_\delta = \lfloor \lfloor \ldots \lfloor vx \rfloor_{s_1} \ldots \rfloor_{s_{n-1}} \rfloor_{s_n}$. Note that the order of selection is irrelevant. We employ the variational type constructor V to distinguish variational values from non-variational ones. For example, we write $3 : Int$ for plain integers and $A\langle 1, 2\rangle : V(Int)$ for variational ones.

[2]The backquotes allow the writing of binary function as infix operators.

A. Variational Pictures

One application of the choice calculus is for the representation of variational pictures [9]. Variational pictures are structures that encode arbitrarily many different pixel-based pictures. If a traditional picture is modeled as a function from pixel grid locations to T values (often colors), $Pic_T = Loc \to T$, then we can understand a variational picture to be a function form pixel grid locations to variational T values.

$$VPic_T = Loc \to V(T)$$

This allows us to define variational pictures by wrapping the pixels that vary in choices. For example, we could construct a small four-pixel variational picture $v = \begin{smallmatrix} \circ & A\langle B\langle \bullet, \bigstar\rangle, \circ\rangle \\ \bullet & A\langle \circ, \circ\rangle \end{smallmatrix}$.

The two left pixels do not vary, while the two right pixels do. The top-right pixel varies in both the A and B dimensions, while the bottom-right pixel varies only in the A dimension. Because we have two dimensions we can be selected independently, this variational picture encodes four variant pictures (see below). The semantics of variational pictures is a mapping from decisions to plain pictures.

$$[\![\cdot]\!] : VPic \to V(Pic)$$
$$[\![vp]\!] = \{(\delta, (l, x)) \mid (l, vx) \in vp, (\delta, x) \in vx\}$$

With this definition we can get the variants of the variational picture v as $\{(\{A.l, B.l\}, \begin{smallmatrix} \circ & \bullet \\ \bullet & \circ \end{smallmatrix}), (\{A.l, B.r\}, \begin{smallmatrix} \circ & \bigstar \\ \bullet & \circ \end{smallmatrix}), (\{A.r\}, \begin{smallmatrix} \circ & \circ \\ \bullet & \circ \end{smallmatrix}), \}$.

We employ the idea behind variational pictures as a model for representing visualizations that include variation. An important difference is that we do not represent variational pixels but apply variation to basic and composed visualization objects.

IV. Variational Visualizations

A variational visualization is a data visualization that encodes arbitrarily many different plain visualizations and provides a mechanism to navigate through all of the encoded variants. The differences among the encoded visualizations could be aesthetic such as colors or labels, they could be in terms of how the visual parameters are bound, in terms of the data being visualized, or some combination of these factors. To reason about all these possibilities we need to manage the variation in a systematic way.

A. Understanding Different Variability Designs

The visualization model described in Section II provides only limited support for comparison tasks. In particular, without an explicit representation for variation, the opportunities for navigating and manipulating variational visualizations is rather limited. On the other hand, the model of variational pictures in Section III-A is too limited to handle the transformation of variational visualizations. For that, we need yet another application of the core ideas in the choice calculus.

To illustrate this, consider using the variational type constructor V, introduced in Section III, to make arbitrary types variational. It might look something like this.

$$\begin{array}{rcl} V(a) & ::= & D\langle V(a), V(a)\rangle \\ & \mid & a \end{array}$$

A value of type $V(a)$ is either a plain value, or a choice of two nested $V(a)$ values. A $V(a)$ value is a binary tree where the nodes are choices and the leaves are values of type a. This definition allows the top-level application of V to the visualization type defined in Section II, that is, a variational visualization would have the type $V(Visualization)$. For example, We can construct the variational chart PickColor$\langle blueChart, greenChart\rangle$, which allows the selection between two charts as a whole, but it does not support comparing parts of two visualizations in context. This can be important if two visualizations are similar overall but differ only in a few places.

In addition to this top-level application of variability, there are two other possibilities to integrate V into structures [10], namely at the leaves or recursively.

Application of the V type constructor at the leaves involves moving the variation into the visualization structure, applying it directly to the marks. We could redefine our visualization type to be the following.

$$\begin{array}{rcl} Visualization & ::= & V(Mark) \\ & \mid & \textbf{NextTo}\ Visualization^* \\ & \mid & \ldots \end{array}$$

Since we likely want to avoid constructing all of our variational marks manually, we would need to update many of the built-in operations such as *barchart*. In order to construct a variational visualization where the marks are the parts that vary, we need to provide *barchart* with variational data as input. That is, instead of the type

$$barchart : List(Number) \to Visualization$$

we would have something of the form

$$barchart : V(List(Number)) \to Visualization$$

This new *barchart* function would then be responsible for creating a mark for each variant data value.

By pushing the type constructor down into the marks we are able to eliminate two of the major drawbacks from the top-level approach: First, redundant visualization structures can be avoided because the variation can only occur at the innermost level, and second, we can also determine exactly where the variation occurs by observing the marks and their variation directly.

However, the leaf-level application of V prevents many kinds of useful variations in visualizations. For example, we are not able to represent a bar chart that is either a vertically or a horizontally oriented bar chart depending on the selection. These have subtly different structures (**Above** as opposed to **NextTo**), and since we only allow marks to vary, and not the composition operators, this is not possible. Therefore, this approach is obviously too limited to be useful in the general case.

A final possibility is to integrate the variability directly into the recursive structure of the visualizations, allowing it to occur wherever is most appropriate for the desired effect. We can add a new case to the visualization definition as follows.

$$\begin{array}{rcl} Visualization & ::= & \ldots \\ & \mid & V(Visualization) \end{array}$$

The added flexibility of this recursive application of V allows us to avoid any issue with being unable to represent particular kinds of variation. Moreover, assuming the variation is allocated judiciously, we can also avoid unnecessary redundancy.

However, the burden of choosing where to integrate variation is shifted onto the visualization author. Given such a system, the user must have a sufficient understanding of its inner workings to not only know when it is preferable to move the variability inward or outward in the visualization structure, but also how to actually achieve this by using and defining operations. Still, given the the drawbacks of the top-level and leaf-level approaches, the additional demands on the user seem to be reasonable.

B. Rendering and Navigating Variational Visualizations

Having found a suitable way to integrate variability into our visualizations, we still need to decide how they can be presented to and navigated by users. We have implemented a prototype, which renders variational visualizations according to the model of variational pictures described in Section III-A, that is, it produces a variational picture where each variant plain picture is one of the variant plain visualizations. All of the figures used in this paper have been generated by this prototype.

A tool for displaying variational visualizations must allow users to navigate among the different variants. For simplicity, we chose a simple approach based on standard GUI elements. We extract all of the dimensions that a variational visualization contains and produce a checkbox for each. This checkbox toggles the selection in that dimension. When one of the dimensions is toggled on, radio buttons are shown to select between either the left or right alternatives. This scheme allows users to specify any decision, whether partial or total, and see the rendered result.

When a decision is not total, and variation still exists in the visualization being rendered, it is not obvious what should be drawn. One possibility is to draw nothing for the parts that are unselected and just indicate that a selection must be made first, such as by a box or outline. Since we are focusing on comparison tasks, however, we chose an approach in which all variants of the current visualization are shown at once, side-by-side, using a small multiples approach. This not only supports comparison but also gives users the ability to see visually how much variation remains unselected in their visualization.

We have also used colors to map the navigation interface to the portions of the rendering canvas that they affect. For example, if a particular dimension toggles the height of a bar, we outline that bar's space with a colored, dashed outline and color the corresponding UI elements with the same color. These colors are assigned automatically. A screenshot of the prototype is shown in Figure 3.

C. Variational Comparative Visualizations

The simplest example of a variational visualization is to combine two existing visualizations in a choice, as indicated above with the choice between the green and blue barchart.

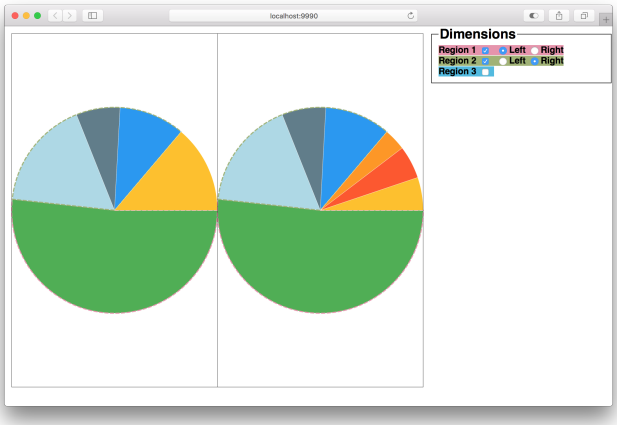


Fig. 3: A screen capture of our prototype user interface showing possible configuration/selection options. On the left is the rendered visualization currently being constructed and on the right are the interface elements which allow the user to navigate among the variants.

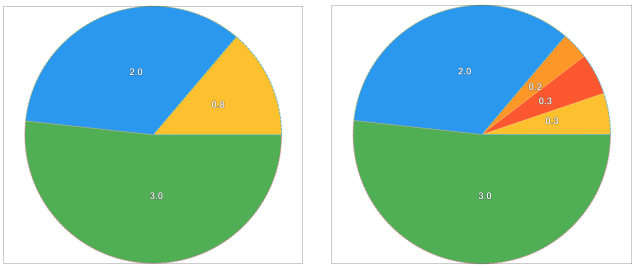
Recall from Section III that choice expressions can be simplified by selection. However, if the selection is performed with a decision that is not total, i.e., that does not map every choice to a particular alternative, then the variation is not entirely removed. In our prototype, when variation is not entirely eliminated with selection, we render all of the remaining variants in a small multiples grid.

Another use of variation is to control the level of detail shown for a set of data. For example, suppose we want to produce a pie chart showing a breakdown of some costs for geographic regions in the United States. We might want to show an overview for areas such as West coast, East coast, South, etc. (Figure 4a). However, we also want to make the details of the states comprising each region available on demand by selecting corresponding variants (Figure 4b). We can encode the zoomed out and zoomed in versions of each pie wedge in a choice, allowing variation to take care of the exponentially many different versions.[3] The rendered output is shown in Figure 4.

$$\begin{aligned}\textbf{Polar}\ (\textbf{NextTo}\ [&\mathit{West}\langle 2.0, \textbf{NextTo}\ [1.0, 0.4, 0.6]\rangle\\ &\mathit{East}\langle 0.8, \textbf{NextTo}\ [0.2, 0.3, 0.3]\rangle\\ &\mathit{South}\langle 3.0, \textbf{NextTo}\ [1.9, 0.7, 0.4]\rangle)\end{aligned}$$

Variation is not just useful for comparing aesthetic options, however. One of the major advantages that an approach based on variation gives us is the ability to work directly with variational data. Suppose we want to examine source code that is annotated with C-preprocessor macros such as `#ifdef` to chart the number of lines of code in each block. We could produce each of the 2^n possible configurations (where n is the number of configuration options), count the relevant lines of code in each, and then produce a separate chart for each configuration.

[3]Some details regarding how the relative widths of nested visualization components are computed and the color assignment are omitted here for simplicity.



(a) (b)

Fig. 4: Comparative visualizations for exploring visualization details; (a) Summary visualization that corresponds to the decision $\{West.l, East.l, South.l\}$; (b) Revealing details for the east with decision $\{West.l, East.r, South.l\}$.

This, however, is generally infeasible since large software projects have hundreds or even thousands of configuration options [11]. However, if we instead count the number of lines directly, but encode the values as variational numbers corresponding to the preprocessor macros, we can perform all of the work in a single pass. Since the data and the visualization tool would be making use of the same variation representation, we can then just chart the results directly. For example, we could make use of the *vbarchart* function, which is the variational equivalent of *barchart*.

$$vbarchart : V(Number) \rightarrow Visualization$$

The viewer of the visualization can then navigate the different configurations to compare the results for each.

Finally, we can compute modified versions of visualizations to compare between. Suppose, for example, we have two bar charts showing monthly earnings over the past two years for a business. We would now like to compare the two years directly by charting the change from the first year to the second for each month. We can use the *zipWith* function, which accepts two visualizations as input, traverses them in parallel, and computes a new visualization element based on a binary operation. In this example, we want to subtract the height of the bars in the second chart from the height of the bars in the first. We can simply use *zipWith* together with the (-) function, which determines which visual parameter(s) are bound to data and computes their differences. For cases in which several visual parameters are bound to data, users can specify exactly which should be acted upon.

$$zipWith\ (\text{-})\ bar_1\ bar_2$$

An example of this computed visualization, composed next to the two original charts, is shown in Figure 5a.

Similarly, we can also compute data transformations directly on visualizations. For example, we may have a set of data already charted for which we also want to see both the log transformed version and a square root transformed version. Moreover, we want to see the original and transformed data at the same time, overlaid using transparency. We can achieve the result shown in Figure 5b in the following way. Note that the figure shows the output when we have not selected either alternative, meaning both are shown using a small multiples approach.

$$\begin{aligned}\textsc{MapType}\langle&\textbf{Overlay}[bar_1\ \text{`}alpha\text{`}\ 0.5, map\ log\ bar_1]\\ &\textbf{Overlay}[bar_1\ \text{`}alpha\text{`}\ 0.5, map\ sqrt\ bar_1]\rangle\end{aligned}$$

Here we are using the *map* function to apply a transformation directly to the visualization elements rather than first transforming the data and only then creating a new visualization.

Finally, we can also perform computations across entire visualizations, such as when sorting elements. Perhaps, for example, we have created some donut charts and realize now that they may be easier to read when sorted. Again, we can avoid having to copy and paste code or start from scratch by directly sorting the elements of an existing visualization.

$$\begin{aligned}\textsc{Sorted}\langle&\textbf{Polar}\ (\textbf{Above}\ [pie_1, pie_2]),\\ &\textbf{Polar}\ (\textbf{Above}\ [sort\ pie_1,\ sort\ pie_2])\rangle\end{aligned}$$

Figure 5c shows the result.

V. Evaluation of Variation for Comparison

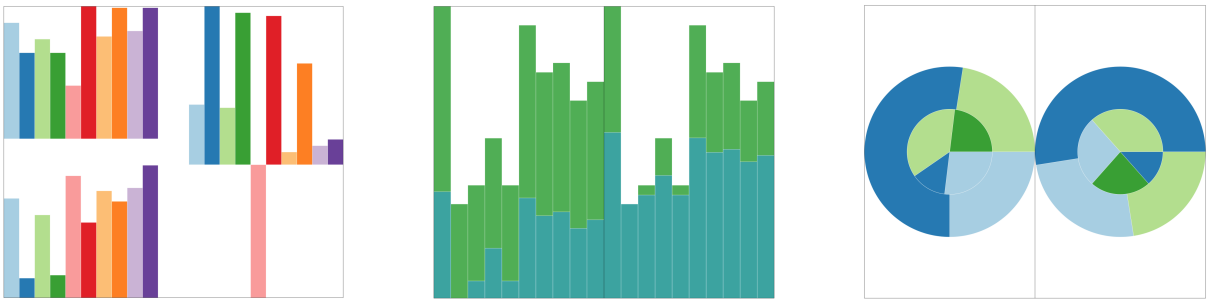
To evaluate how well variation and parameterization is able to serve as a model for comparative visualization, we need to know what features are required. Gleicher et al. [12] propose a taxonomy of visual designs used for such comparison tasks. The taxonomy is validated through a significant survey of work.

Their taxonomy of comparative designs categorizes all of the work surveyed into three main categories (as well as pairs of categories) *juxtaposition*, *superposition*, and *explicit representation of the relationships*. Additionally, there are hybrid categories which combine two of these approaches into one design. We explore each of these options in turn and demonstrate which parts of our model can be used to express them.

A. Juxtaposition

The core idea of juxtaposition is to support comparison tasks by placing the objects to be compared into separate spaces. The objects are always shown independently and in their entirety. One common form is spatial juxtaposition, also often called small multiples, in which the objects to be compared are all shown and arranged (often as a grid) in the available space. The taxonomy also allows for juxtaposition in time, in which objects are displayed one after another in sequence.

Our model of variational visualizations supports juxtaposition in more than one way. The easiest way to achieve it is to use variation to encode the visualizations we want to compare, and then rely on the default behavior of our prototype tool. This renders a small multiples view of all visualization alternatives that are not explicitly selected. Another approach is to use the spatial composition operators explicitly, such as **Above** and **NextTo**. These juxtapose visualizations geometrically by dividing up the available space equally. However, plain spatial composition does not support the selective display and navigation of alternatives provided by variations.



(a) (b) (c)

Fig. 5: Examples of comparative visualizations using hybrid designs. In (a) we have two charts on the left which are zipped together to produce the chart on the right by subtracting the heights of the lower chart from the top one. Note that each chart is scaled independently and so simply measuring the bars would be misleading. Figure (b) shows a small multiples rendering of a data set overlaid with its log transformed data on the left and square root transformed data on the right. Finally, (c) shows a variational visualization in which the left variant shows the original data and the right variant shows the visualizations sorted after their creation.

Finally, juxtaposition can also be temporal, as in an animation that cycles through a set of visualizations. Our prototype does not support animation directly, but it is trivial to replicate this behavior using choices. We simply encode the variants as part of a variational visualization, as in the first example, and map each step of the animation to a particular selection. It is then easy to envision a tool which allows the user to define an animation by setting a timer which navigates among the desired selections.

Therefore, we can say that juxtaposition can be modeled by a combination of variation and spatial composition.

$$\textit{Juxtaposition} \approx V \oplus \textit{SpatialComposition}$$

B. Superposition

The superposition category includes designs in which the objects being compared all share a single space. In general, this is realized by composing visualizations via an overlay operation. Aesthetic tweaks such as transparency and small shifts to avoid totally obscuring some objects are common. Superposition also frequently requires some computation to determine an alignment for different objects. In Section IV-C we showed examples making use of overlays, transparency, and spacing directly.

Because our model offers the ability to define functions over visualizations, realizing a general mechanism for parameterization, we can employ computation at essentially all levels. We have shown examples that include sorting values. We can also envision more sophisticated scenarios such as changing the order of overlaid charts or organizing a small multiples layout based on some derived value from a set of charts.

It is clear from these examples that superposition can be modeled by parameterization/computation and **Overlay**.

$$\textit{Superposition} \approx \textit{Parameterization} \oplus \textbf{Overlay}$$

C. Explicit Representation of Relationships

The final category of the taxonomy includes designs which encode the relationships among the objects being visualized directly. One example we have already seen of this is charting the difference between two ordered data sets (using *zipWith*) and visualizing that result rather than showing both original data sets. A design in this category always involves the extra step of computing the relationships among objects before anything can be visualized. In general, visualizations following these design principles do not require variation techniques in our model, since we have access to computation. We have also shown how we can apply data transformations to individual visualizations, including log and square root transformations.

Finally, in some cases variation can also be used to explicitly encode relationships. One example of this is the pie chart in Section IV-C in which the variation controls the visible level of detail.

Based on these examples, we see that the explicit encoding approach can be modeled by a combination of variation and parameterization.

$$\textit{ExplicitEncoding} \approx \textit{Parameterization} \oplus V$$

D. Hybrid Categories

The taxonomy also includes designs that take hybrid approaches. Combining the three original categories into pairs results in three new hybrid categories, juxtaposition and superposition, juxtaposition and explicit encoding, and superposition and explicit encoding. Each of these is manifested in designs included in the survey and so are necessary to include.

Due to our compositional design of visualizations, all of the functionality discussed so far is essentially orthogonal and all techniques can be composed. For example, to support juxtaposition and superposition at the same time, we can use any of the approaches mentioned above in Section V-B to produce visualizations making use of superposition. With those results, we can then compose those charts together (using language constructs or variation, as described in Section V-A) to produce a hybrid visualization.

Analogously, for juxtaposition and explicit encoding we can apply any desired computations to explicitly visualize relationships among objects and data and then compose

those into larger, hybrid visualizations. One example of this would be to juxtapose (using variation) two charts which are themselves variational, as we did with the log and square root transformation example in Section IV-C.

Finally, for a hybrid approach involving superposition and explicit encoding we can compute any desired relationships among objects and then add them to the overlay composition used in superposition. Conveniently, the same log and square root transformation example also demonstrates an example of this category.

E. Evaluation Conclusions

Since our model is intentionally limited to a small subset of visualization types we do not claim to be able to reproduce most of the actual designs surveyed to produce the taxonomy. However, we have shown how an approach based on parameterization and variation can, in principle, support any combination of identified comparative designs (see the summary in the following table).

	V	*Parameterization*	*Spatial Composition*	*Overlay*
Juxtaposition	×		×	
Superposition		×		×
Explicit Repr.	×	×		

Our prototype implementation is able to handle all of the core ideas underlying the comparative designs.

VI. Related Work

There is no shortage of visualization tools and models, and it is beyond the scope of this section to characterize them all. We therefore mention only on those which directly influenced this work. While D3 [13] has since supplanted it and gained widespread adoption, its predecessor Protovis [14] is closer in design to our model. Protovis was based on a declarative domain-specific language which separated the specification of visualizations from the rendering process [15]. Protovis was superseded by D3 primarily because the authors aspired to create a tool for web developers to be able to do more than creating visualizations. D3 is less a visualization tool than a library for data-driven web content. The cost of this added flexibility is the elimination of domain-specific constructs. The change seems to have been motivated by a rethinking of the goals of the project. We believe that the domain-specific approach still has value for many users, which is witnessed to some extent by the creation of many libraries in the community that abstract over D3.

Both ggplot2 [16] and the grammar of graphics which underpins it [17] serve as inspiration particularly for visualization transformations, although it is not designed to support programmability and is therefore generally fixed in what operations are supported.

The Haskell domain-specific language Diagrams, described partially by Yorgey [18], supports the creation of diagrams through composition and relatively spacing and directly informs many of the concepts used to define our model of visualizations. Diagrams does not directly support data-driven graphics and so is not suitable for a general purpose visualization tool.

Comparative visualization is a large and growing field. Over roughly the past two decades, a number of works in information and scientific visualization have advocated for and distinguished deliberate visual comparison designs, including Pagendarm and Post [19], Woodring and Shen [20], and Roberts [21]. Naturally Gleicher et al. [12], referred to throughout this work, provides a thorough overview of the field as well as a taxonomy of comparative designs.

Comparison tasks are also a core part of visualization history tools. Interacting with the history of a user-created visualization artifact is itself too broad of a subject to fully summarize here, so we refer to Heer et al. [2] which studies and organizes the design space of graphical history tools.

Finally, another area in which visual comparison tasks arrive routinely is in uncertainty visualization. Uncertain or missing data often lead naturally to a large, or even infinite set of possible visual representations. One example is weather forecasting with uncertain parameters, which can result in needing to compare a collection of different results, as described by Bonneau et al. [3]. That work also effectively summarizes sources of uncertain data as well as current approaches and unsolved problems.

To our knowledge, no work has yet tried to apply a systematic model of variation explicitly to support visual comparison tasks. However, some work makes implicit use of comparison for variation-based exploratory tasks. For example, Side Views [22] and Parallel Paths [23] designed live "what-if" previews for graphical operations which implicitly relies on comparison. Hartmann et al. [24] took a variation-based approach to user interfaces and interactions which require comparison tasks. As mentioned, the work on variational pictures [9] makes use of variational area trees to help support comparison and exploration tasks.

VII. Conclusions & Future Work

We have shown an approach to information visualization based on parameterization and variability. Through examples, we have demonstrated the suitability of this approach for creating not only visualizations in general, but specifically those that support visual comparison tasks.

We have evaluated our model by showing how it is able to instantiate visualizations in every category of Gleicher's taxonomy of comparative designs [12]. Accordingly, we posit that parameterized variational visualizations offer an effective model of comparative visualization more generally.

In future work, we will extend the implementation of our visualization DSL with general control structures and operators to introduce, maintain, and consume variational visualizations. This will offer users an exploratory approach to information visualization in general.

Acknowledgements

This work is partially supported by the National Science Foundation under the grants IIS-1314384 and CCF-1717300.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The evolution of data to life-critical," IDC, Tech. Rep., 2017.

[2] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala, "Graphical histories for visualization: Supporting analysis, communication, and evaluation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1189–1196, 2008.

[3] G.-P. Bonneau, H.-C. Hege, C. R. Johnson, M. M. Oliveira, K. Potter, P. Rheingans, and T. Schultz, "Overview and state-of-the-art of uncertainty visualization," in *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*, C. D. Hansen, M. Chen, C. R. Johnson, A. E. Kaufman, and H. Hagen, Eds. Springer London, 2014, pp. 3–27.

[4] D. Holten and J. J. van Wijk, "Visual comparison of hierarchically organized data," in *Joint Eurographics / IEEE VGTC Conference on Visualization*, ser. EuroVis '08, 2008, pp. 759–766.

[5] E. R. Tufte, *The Visual Display of Quantitative Information*, 2nd ed. Graphics Press LLC, 2001.

[6] K. Smeltzer, M. Erwig, and R. Metoyer, "A transformational approach to data visualization," in *International Conference on Generative Programming: Concepts and Experiences*, 2014, pp. 53–62.

[7] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps*. Morgan Kaufmann Publishers Inc., 1999, English translation.

[8] M. Erwig and E. Walkingshaw, "The choice calculus: A representation for software variation," *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 1, pp. 6:1–6:27, 2011.

[9] M. Erwig and K. Smeltzer, "Variational Pictures," in *Int. Conf. on the Theory and Application of Diagrams*, ser. LNAI 10871, 2018, pp. 55–70.

[10] K. Smeltzer and M. Erwig, "Variational lists: Comparisons and design guidelines," in *ACM SIGPLAN International Workshop on Feature-Oriented Software Development*, 2017, pp. 31–40.

[11] J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze, "An analysis of the variability in forty preprocessor-based software product lines," in *ACM/IEEE International Conference on Software Engineering*, 2010, pp. 105–114.

[12] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts, "Visual comparison for information visualization," *Information Visualization*, vol. 10, no. 4, pp. 289–309, 2011.

[13] M. Bostock, V. Ogievetsky, and J. Heer, "$\mathbb{D}^3$: Data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[14] M. Bostock and J. Heer, "Protovis: A graphical toolkit for visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1121–1128, 2009.

[15] J. Heer and M. Bostock, "Declarative language design for interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1149–1156, 2010.

[16] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, 2nd ed. Springer, 2016.

[17] L. Wilkinson, *The Grammar of Graphics*. Springer Science & Business Media, 2006.

[18] B. A. Yorgey, "Monoids: Theme and variations (functional pearl)," in *Haskell Symposium*, 2012, pp. 105–116.

[19] H.-G. Pagendarm and F. H. Post, "Comparative visualization: Approaches and examples," in *Visualization in Scientific Computing*, M. G. andâĂŐ Heinrich Müller andâĂŐ Bodo Urban, Ed. Springer-Verlag, 1995.

[20] J. Woodring and H.-W. Shen, "Multi-variate, time varying, and comparative visualization with contextual cues," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 909–916, 2006.

[21] J. C. Roberts, "State of the art: Coordinated multiple views in exploratory visualization," in *International Conference on Coordinated and Multiple Views in Exploratory Visualization*, 2007, pp. 61–71.

[22] M. Terry and E. D. Mynatt, "Side views: Persistent, on-demand previews for open-ended tasks," in *ACM Symp. on User Interface Soft. and Tech.*, 2002, pp. 71–80.

[23] M. Terry, E. D. Mynatt, K. Nakakoji, and Y. Yamamoto, "Variation in element and action: Supporting simultaneous development of alternative solutions," in *SIGCHI Conf. on Human Factors in Comp. Systems*, 2004, pp. 711–718.

[24] B. Hartmann, L. Yu, A. Allison, Y. Yang, and S. R. Klemmer, "Design as exploration: Creating interface alternatives through parallel authoring and runtime tuning," in *ACM Symposium on User Interface Software and Technology*, 2008, pp. 91–100.