Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring

Zhicheng Liu[1]   John Thompson[2]   Alan Wilson[3]   Mira Dontcheva[1]
James Delorey[3]   Sam Grigg[3]   Bernard Kerr[3]   John Stasko[2]

[1]Adobe Research
Seattle, WA
{leoli,mirad}@adobe.com

[2]Georgia Institute of Technology
Atlanta, GA
{jrthompson,stasko}@gatech.edu

[3]Adobe Systems Inc.
Lehi, UT & San Francisco, CA
{alawilso,delorey,grigg,bkerr}@adobe.com

ABSTRACT

Building graphical user interfaces for visualization authoring is challenging as one must reconcile the tension between flexible graphics manipulation and procedural visualization generation based on a graphical grammar or declarative languages. To better support designers' workflows and practices, we propose Data Illustrator, a novel visualization framework. In our approach, all visualizations are initially vector graphics; data binding is applied when necessary and only constrains interactive manipulation to that data bound property. The framework augments graphic design tools with new concepts and operators, and describes the structure and generation of a variety of visualizations. Based on the framework, we design and implement a visualization authoring system. The system extends interaction techniques in modern vector design tools for direct manipulation of visualization configurations and parameters. We demonstrate the expressive power of our approach through a variety of examples. A qualitative study shows that designers can use our framework to compose visualizations.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): UI

Author Keywords

Data visualization; graphic design; interaction techniques; framework; authoring; systems.

INTRODUCTION

Graphic designers have been producing infographics and charts well before the recent proliferation of computer generated visualizations [20, 37]. As visualization becomes an increasingly popular medium for storytelling and communication, there is a renewed and growing interest to understand visualization creation from the perspective of graphic design [3, 4, 32, 56, 57]. Prior studies show that graphic designers approach visualization authoring differently from computer scientists: they often start by thinking about the high-level

appearance of a visualization in terms of layout and space configuration, and focus on encoding real data into the visuals later [3, 56]. The discipline of graphic design has also established a rich set of concepts and tools that are widely used in the community. For example, professional vector editors enable designers to work with shape geometries at the level of anchor points and curve segments. The grid system and smart guides serve as two powerful tools to precisely structure visual elements and configure display space [32, 40, 56].

Despite the plethora of existing visualization creation tools, few tried to incorporate designers' workflow and practices into system and interface design. Vuillemot and Boy [56] argue that most visualization tools follow a bottom-up, data-to-graphics process as described in the information visualization (InfoVis) reference model [8]: starting with data, one performs data transformation, visual mapping, and view transformation to generate visualizations. This model informed the development of powerful visualization algebra and declarative languages [29, 49, 52, 58]. However, these tools often require coding expertise, or are not flexible enough for design practices.

Systems like Lyra [48] and iVisDesigner [44] offer graphical user interfaces (GUI) for visualization authoring without programming, thus are more flexible. These efforts start with template or grammar-based visualization generation engines, and design interfaces for changing generative parameters. Such approaches still need to reconcile the potential tension between flexible change of graphical configurations and the formalism imposed by generation engines [4]. To bridge the gap between generation engines and drawing tools, Hanpuku [4] implements a streamlined model for visualization authoring across multiple tools.

Recent work also began to explore visualization authoring without programming from a purely graphic design perspective. With Data-Driven Guides [32], designers can create freeform guides and sketch graphics with the guides. d3-gridding [56] enables the creation of quick mock-ups with minimal or no data. These systems adopt a "lazy data binding" approach: visualizations are first and foremost vector graphics with no underlying templates or declarative languages. Designers use familiar tools to draw, select and manipulate vector graphics, and apply data encoding only when it is necessary. Compared to template or grammar based systems, this approach is more compatible with designers' workflows and practices. Users do not have to align their mental models with

the grammar or model assumed by the system. Furthermore, vector design tools are highly flexible and expressive: with enough time and patience, one can create virtually any graphics. Augmenting these tools with data encoding support can reduce manual effort without disrupting designers' workflows.

The lazy data binding approach is promising, but needs to be developed further to support a wide variety of visualizations. Data-Driven Guides [32] only focus on infographics with simple layouts. d3-gridding [56] primarily supports design mockups, and still requires programming. It remains a challenge for designers to create high-fidelity data visualizations with complex visual mappings and layouts.

Consider the visualizations in Figure 1: Figure 1(a) is a slope graph used on the cover of Alberto Cairo's book *The Functional Art* [7], showing U.S. states' obesity and education percentages (hereafter referred as the "Obesity vs. Education" visualization); Figure 1(b) visualizes the NBA draft over the past 20 years (x axis) and the order of players in terms of draft pick (y axis) [15] (hereafter referred as the "NBA Redraft" visualization); Figure 1(c) is a multi-series line graph visualizing four companies' monthly stock prices [36] (hereafter referred as the "Stock Prices" visualization); Figure 1(d) is "A Field Guide to Red and Blue America" by Wall Street Journal, showing the PVI (Partisan Voter Index) for each state over the past 9 elections [61] (hereafter referred as the "Red and Blue America" visualization). Each small bar chart represents a state, and is positioned according to US geography.

Designers might be able to use existing drawing tools or Data-Driven Guides to create these examples, but the process will be painful. Generating shapes or points on lines (Figure 1(c)) can be tedious and slow; organizing the shapes into meaningful layouts (Figure 1(b) and (d)) and map data to positions and color (Figure 1(c)) are daunting manual tasks. To enable designers to keep using the powerful drawing tools and to automate the repetitive work, we need a systematic framework with sufficient descriptive and generative power.

In this paper we propose a novel framework for visualization authoring based on the lazy data encoding approach. This framework describes components in a visualization using graphic design concepts such as *shape*, *anchor point*, *segment*, and *group*. Two operators, *repeat* and *partition*, generate shapes and anchor points, and attach data to them. The resultant visual components each has a *data scope*, and are considered *peers* of each other inside a *collection*. Collections use *layouts* to arrange shapes, and can be *nested* to create more complex organizations. Data serves as *constraints* when bound to visual properties, and unbound properties can be freely manipulated. These components and operators can describe the structure and generation of a wide range of visualizations.

Informed by the framework, we design and implement the Data Illustrator system. We augment interactive techniques in modern vector design tools for direct manipulation of visualization configurations and parameters. We demonstrate the expressive power of our approach through a range of examples. To better understand the strengths and limitations of our approach, we conduct a qualitative user study with 13

(b) NBA Redraft

(c) Stock Prices

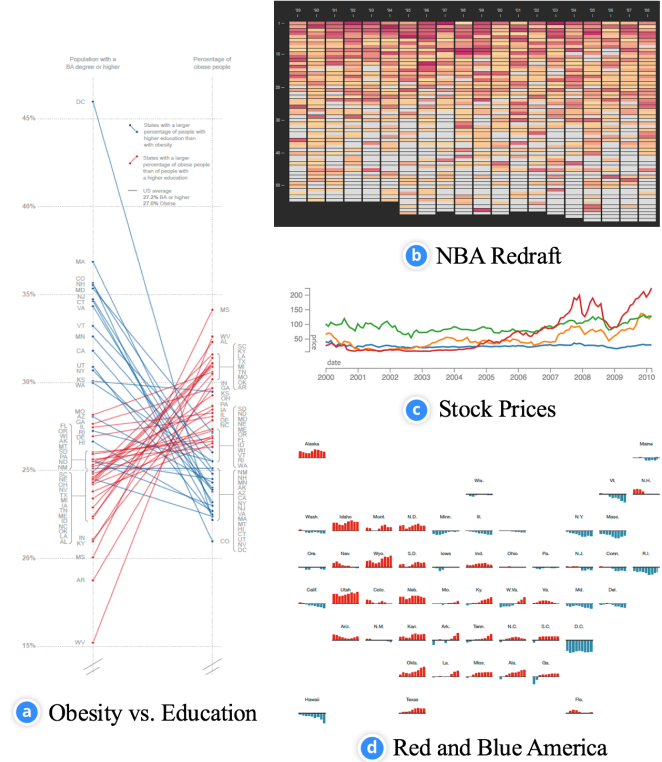(a) Obesity vs. Education

(d) Red and Blue America

Figure 1: Visualizations with varying levels of complexity

designers, focusing on whether they can understand and use the framework for visualization composition.

FORMATIVE STUDY AND DESIGN ITERATIONS

To understand how different visualizations could be described and created from a graphic design perspective, we held one-hour weekly meetings with three designers over a period of two years. All three designers have more than ten years of experience in graphic design, digital illustration, web design and print design. Two of the designers have also created infographics and data visualizations on a regular basis as part of their work. The designers frequently used applications such as Adobe Photoshop [25], Illustrator [23], InDesign [24] and Experience Design (XD) [22], Sketch [10], and Figma [12]. These applications represent the industry standard for design professionals. They share a similar set of features and tools, varying in terms of interaction and interface design.

In the initial meetings, we collected visualizations by sampling chart types from systems like Tableau [51] and stylistic information graphics from websites such as "the Kantar Information is Beautiful Awards" [31]. Each week we asked the designers to describe at a high level how they would create one of these visualizations and demonstrate the workflow using the tools of their choice. We told the designers to assume that the system would take care of data binding automatically. These exercises helped us understand designers' way of thinking and workflow through concrete examples, and familiarized ourselves with an assortment of professional design tools.

Three main tasks are key in visualization authoring. Designers performed the following three tasks for all the examples: (1) sketch and generate shapes, which were accomplished by drawing tools (e.g. *Pen Tool* in Adobe XD) and duplication tools (e.g. *Copy & Paste*), (2) arrange and organize shapes, where grids and groups were extensively used, and (3) bind data to visual properties, which was not supported in most design tools. Two pieces of insight were consistent with previous findings [53], and directly informed the focus on *repeat* and *layout* in our framework: repeated shapes were dominant in data visualizations, and position encoding were often relational instead of data-driven (i.e. a shape's position depended on the placement of related shapes).

Workflow is largely top-down, but data is not always an afterthought. Observations from these exercises confirmed our intuition and previous findings [3, 56] that designers think about graphical aspects of visualizations before data encoding. However, the authoring processes were not strictly divided into a visual design stage followed by a data encoding stage. When drawing and manipulating shapes, sometimes it was beneficial to bring in real data. For example, one designer showed how he would use the *Blend Tool* [28] in Illustrator to create multiple copies of a shape. He first drew two shapes on canvas, and then used the *Blend Tool* to interpolate a predefined number of shapes between them. Instead of having to define an arbitrary number, the designer wanted automatic generation of the number based on real data.

Direct manipulation enhances flexibility and reduces semantic distance. Designers treat the canvas not only as a scene for production, but also a playground for experimenting with ideas. It is important that they can flexibly and quickly sketch shapes, and change visual configurations or appearances with full control and precision. All the designers greatly valued direct manipulation features that gave them immediate visual feedback. Simple operations such as dragging corners to resize a shape, or dragging shapes to move them around can be immensely useful. Such features are commonplace in design tools, but are rarely supported in visualization systems.

In the second phase of the formative study, we explored how existing design tools could be directly used or augmented to support the three main tasks. We conducted weekly design meetings for 15 months. Each week we created a storyboard to illustrate step-by-step visualization authoring scenarios. In total we produced about 40 design sketches and mockups. For instance, we spent one month brainstorming how to augment existing design tools to create visualizations in the line graph category. In a line graph, one polyline plots all the data, and the points on the line represent individual data cases. Grammar-based approaches solve this problem through declarative specification (e.g. *line*(*position*(date ∗ value)) [60]) . To designers, however, such specification did not make sense because a line's position refers to the coordinates of its bounds. Taking a graphics-centric approach, we created storyboards based on different ideas such as repeating points along a line, duplicating a point multiple times and connecting the dots, and dividing a line into segments. We then collected designers' feedback and eliminated ideas that sounded bizarre to them.


Line	Path	Rectangle	Circle
2 anchor points 1 line segment open	4 anchor points 3 line segments open	4 anchor pointss 4 line segments closed	4 anchor points 4 curve segments closed

Table 1: Anatomy of Shapes: Anchor Points and Segments

A great challenge we faced was to construct a coherent set of concepts and tools that behave consistently for diverse visualizations. Often an idea seemed feasible for one visualization design, but turned out to be inconsistent with new examples. In addition to a single line graph, we needed to consider more complex cases such as small multiples of line graphs or multiple lines in a single chart. Moreover, slope graphs and parallel coordinate plots also use line as a visual primitive, but in different ways from a line graph. These diverse visualizations added complexity to the construction of a coherent framework. We kept iterating on the ideas as new use cases arose. After a few months of storyboard creation, we distilled a set of concepts and tools that worked consistently across different visualizations. We then started implementing a prototype which helped further concretize the ideas, and the iterations on framework and interaction designs continued towards the end of the prototyping process.

THE DATA ILLUSTRATOR FRAMEWORK

The principle of consistency underpins the creation of our framework. If we borrow an existing design concept, its meaning and behavior must be consistent with the way it is used in existing design applications. Otherwise we need to devise a new concept. For example, *symbol* is widely adopted in mainstream vector design tools. Users can turn any visual object into a symbol (akin to the concept of class in computer science), and create many instances of the symbol. Changes to the symbol will be propagated to the instances. We tried to use *symbol* to describe the generation of visualizations, but eventually decided that it was a stretch to apply it in the context of visualization authoring. To explain the framework, we use the visualizations in Figure 1 as running examples.

Shapes, Anchor Points and Segments Shapes are the building blocks of visualizations. In professional design tools, shapes are represented as series of anchor points connected by line or curve segments. Table 1 shows a few shape types, with information on the number of anchor points, the number of segments, and whether the path is open or closed. A line is the shape primitive for Figure 1(a), a rectangle for Figure 1(b) and (d), and a polyline/path for Figure 1(c).

Repeat and Partition After sketching a shape primitive, designers can use the repeat and partition operators to generate shapes and attach data to them (Table 2). Repeat creates multiple copies of a shape, and is inspired by duplication tools (e.g. *Repeat Grid* in Adobe XD, *Duplicate* in Sketch); Partition divides a shape into constituent parts, and draws inspirations from the *Knife Tool and Scissors Tool* [26] in Adobe Illustrator.

3

	Repeat	Partition
Concept	creates multiple copies of a shape	divides a shape into constituent parts
Shape	works for all kinds of shape and group	works for lines, rectangles, circles, rings and areas only
Example (line)		
Example (rectangle)		
Example (circle)		

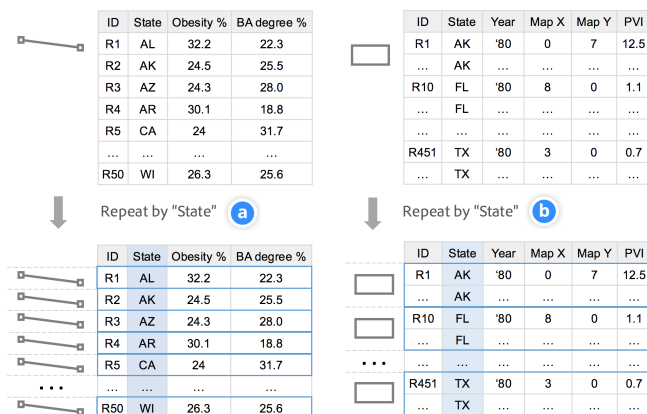Table 2: Repeat vs. Partition

Figure 2: (a) repeat a line by State for "Obesity vs. Education", (b) repeat a rectangle by State for "Red and Blue America"

To create the lines in the "Obesity vs. Education" visualization, we can first repeat a line by State. The repeat operator duplicates the line, and associates each line with a unique State value and the data rows sharing that value (Figure 2(a)). Similarly, we can repeat a rectangle by State (Figure 2(b)) for the "Red and Blue America" visualization. Note that multiple rows share the same State value, and the repeat operator only generates a shape for each unique State value. In general, the data variable used to repeat a shape should be categorical, since the number of repeated shapes must be an integer.

To create the line graphs in the "Stock Prices" visualization, we follow suit and repeat a line by Company (Figure 3(a), top). Next, we partition each line by Date to divide them into multiple line segments. The partition operator generates an anchor point for each Date value, and associates the corresponding data rows with the anchor points (Figure 3(a)). Similarly, to generate an outline for the "NBA Redraft" visualization, we repeat a rectangle by Year first (Figure 3(b), top), then partition the rectangles by Player. In general, the partition operator divides a shape into its constituent parts by a categorical variable. How the division works depends on the shape type, for example, a circle is divided into slices of pie (Table 2).

Data Scope A shape's data scope refers to its attached data rows as a result of the repeat or partition operator. The data scope is usually a subset of the original dataset, described by
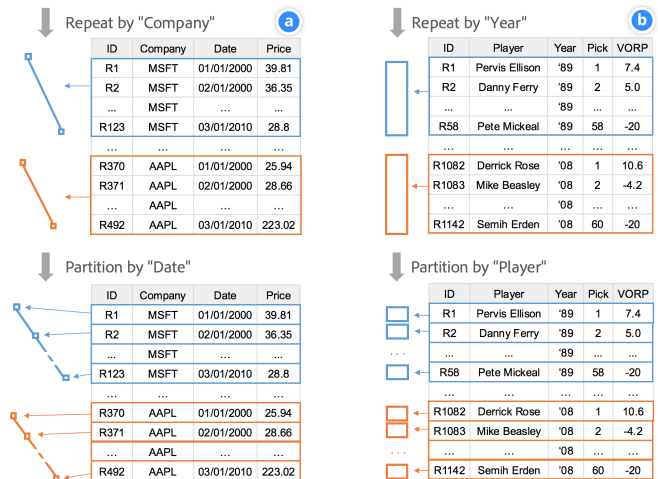

Figure 3: (a) repeat a line by Company then partition lines by Date, (b) repeat a rectangle by Year then partition rectangles by Player

categorical filters. For example, in Figure 3(a), after repeating, the data scope of the first line is the data rows where Company = Microsoft. The anchor points of the line has no data scope yet, only after partitioning, each anchor point has its own data scope: e.g. Company = Microsoft and Date = 01/01/2000. The first filter is inherited from the line's data scope.

Collection vs. Group After repeat (Figure 2), we have a collection of lines or rectangles for each of the four examples in Figure 1. This collection may be considered as a "group": in all the design applications, multiple shapes can be grouped so that they can be moved, scaled or copied at once. In our framework, however, collection and group are two distinct concepts. Table 3 explains the differences between them.

Collection	Group
A collection of polylines, delineated by green dotted borders. The lines are repeated by Company and partitioned by Date (Figure 3(a))	A group of shapes in the "brain drain" visualization [1], delineated by blue borders
The children of a collection have the same type (e.g. rectangle, group)	The children of a group can have different types
Each child inside a collection has a different data scope	All the children inside a group share the same data scope

Table 3: The differences between a collection and a group

Layout To arrange the lines and rectangles in a collection, we apply a layout to the collection. Table 3 shows an example

of a grid layout with one column and four rows. The pink lines are the grid cell boundaries. In general, we can apply the following types of layout to a collection generated by repeat: freeform (i.e. no layout), grid, stack, and packing. Grid layout is an essential tool supported by most design applications. Stack layout exists in fewer applications (e.g. Auto-Layout [2] in Sketch), but is an important feature in visualizations. The main differences between a grid layout and a stack layout include: (1) a grid is two dimensional (rows and columns), a stack is one dimensional (horizontal or vertical); (2) all the grid cells have the same size, and the size of the collection depends on the cell size and number of cells; in a stack layout, the size of the collection is the sum of all the children's sizes. A packing layout is a space-filling arrangement: the layout is equivalent to a Treemap for rectangle shapes, and a packed bubble chart for circles.

Both the grid and stack layouts have a *coordinate space* parameter consisting of two values: Cartesian and Polar. Grid layout in the polar space is inspired by the *Polar Grid Tool* [27] in Adobe Illustrator. Similarly, stack layout has a corresponding representation in the Polar space.

Nested Collection Collections can be nested to create small multiples or visualizations with nested layouts (e.g. stacked bar chart) [30]. We have seen how to create a nested collection in Figure 3(b): first repeat a rectangle by Year to get a collection, then partition the rectangles in the collection by Player. This procedure will generate the structure in the "NBA Redraft" visualization, if we apply a one-row grid layout to the top level collection, and a one-row grid layout to the inner collections obtained from partitioning. Nested collections can also be created by repeating a collection.

Lazy Data Binding as Constraint By default, the lines or rectangles generated by repeat or partition behave as regular vector graphics. Users can select, scale, move, rotate, align, distribute, and delete the shapes. Even after we have organized these shapes in a collection and their positions are constrained by the layout, we can still move the collection as a whole, or edit the anchor points' position and stroke color. Such flexibility allows manual encoding of shape properties, which could be tedious. Automatic data encoding reduces the manual efforts needed, and serves as additional constraints on the manipulability of visual components.

Say we want the stroke color of the four polylines in Table 3 to represent Company to match the "Stock Prices" visualization. We specify a data binding consisting of four parameters: a data variable (Company), a visual property (Stroke Color), a list of visual components([line1, line2, line3, line4]), and an aggregator (e.g. Sum or Mean) if the data variable is numerical and we need to aggregate multiple values. Once applied, the data binding locks the Stroke Color property and prevents it from interactive manipulation. It is still possible to change the range or domain of the scale, which in turn updates the colors. Unconstrained interaction is restored if the data binding is removed.

The data binding operator executes in three steps. First, it computes a list of data values, one per visual component, based on the component's data scope and the aggregator. In the "Stock
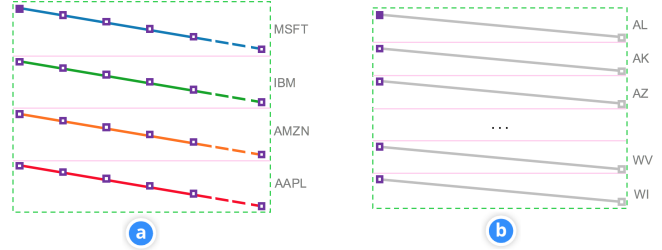

Figure 4: Peers of an anchor point: (a) partitioned polylines, each anchor point has a data scope, (b) paths where the anchor points have no data scopes. The focal anchor point is colored in purple, their peers have purple borders.

Prices" visualization, the data values are the four companies. Second, the binding operator creates a scale. The scale type depends on the data variable type and the visual property (the choice of scale type closely follows the guidelines in D^3 [6]); the scale's domain encompasses the data values computed in the previous step; and the scale's range is determined by the visual property values. Finally, the binding operator transforms the list of data value into property values using the scale, and sets the visual properties. Many systems offer automatic data binding support similar to the description above. Our framework differs in the lazy binding as constraint approach.

Peers Binding Company to Stroke Color results in a unique stroke color for each polyline (Figure 4(a)). Next we need to bind Date to the x position and Price to the y position of the anchor points to create the "Stock Prices" visualization. Here we want the data binding to apply to all the anchor points on all the lines. To create the "Obesity vs. Education" visualization, the data binding works differently. In Figure 2(a) we have repeated a line by State, applying a grid layout to the collection gives us the result in Figure 4(b). We then need to bind BA Degree % to y position of the first anchor point in each line only, and to bind Obesity % to y position of the last anchor points only.

To distinguish these cases and convey how the data binding will work clearly to the users, we introduce the concept of peers. Shapes generated by repeat or partition are peers of each other. For example, the four polylines in Table 3 are peers to each other. What constitutes the peers of an anchor point depends on whether the anchor point has a data scope. When we draw a line and then repeat it by data, the anchor points have no data scopes. The peers of an anchor point are the anchor points at the same index on peer shapes (Figure 4(b)). When we partition a line by data, the anchor points are generated and associated with data. All these anchor points are thus considered peers of each other. If we repeat a partitioned line by data, all the anchor points on all the lines are peers of each other (Figure 4(a)). The concept of peers helps clarifying which visual components should be affected by a data binding.

Layout Taking Precedence over Position Binding The structure enforced by layouts sometimes may be in conflict with binding data to positions. In such cases, the layout takes precedence over position binding. For example, with the four polylines arranged in a grid layout (Figure 4(a)), after binding data to the positions the anchor points, we obtain Figure 5(a).
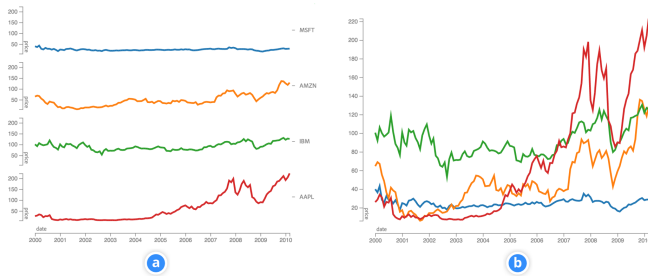
Figure 5: (a) The presence of a grid layout has precedence over position binding, (b) Removing the grid layout unifies the scale of position binding

The position binding only takes effect inside each grid cell. Replacing the grid layout with a freeform layout unifies the scales and axes (Figure 5(b)).

The framework describes the structure and generation of the backbone of the visualizations. In the actual authoring processes, we still need to perform many lower-level tasks, such as configuring the parameters of a layout, ordering and filtering the collection children, and setting the scale range for data binding. In the next section, we discuss the design of the authoring interface based on this framework, so that we can operationalize the framework with flexibility and control.

INTERFACE AND DESIGN RATIONALE

We designed the Data Illustrator application with the following design goals in mind: *familiarity, interpretability, discoverability*, and *control*. Realizing these design goals is a crucial step to ensure that our target audience comprehends and enjoys working with a complex authoring tool. *Familiarity* ensures that the user's previous experience will match their expectations; therefore if our tool uses a feature from an existing vector editing application, we want that feature to be consistent in appearance and behavior. In the case when a novel feature is needed, the design should be *interpretable* by the user. *Interpretability* requires the result of a user action to be immediately comprehended, an important requirement during generative data-bindings. *Discoverability* on the other hand ensures that the interface design shows affordances for interaction so that users can detect the possibility of an action with a visual object. Finally, users need to feel that they are in *control* at every step of the process to realize their design, especially when the system automatically generates aspects of their work.

The interface of Data Illustrator consists of seven components (Figure 6). The Canvas provides space to draw, select and manipulate shapes. The Toolbar on the left contains tools for selecting and drawing shapes - only one can be active at a time. Directly to its right, the Data Variables Panel supports dataset file management. Below that is the Layers Panel which allows users to inspect the canvas. The Actionbar on the far right supports actions for associating data to shapes. Directly below, the Property Inspector displays editable attributes of the currently selected shapes. Finally, the Data Table Panel at the bottom shows all the rows and columns of the dataset and reveals the data scopes of the currently selected shapes.

Figure 6: Seven components of the Data Illustrator interface: 1) Toolbar, 2) Variables Panel, 3) Layers Panel, 4) Canvas, 5) Table Panel, 6) Actionbar, 7) Property Inspector

Drawing Shapes The drawing tools work by click and drag interactions on the canvas. Data Illustrator supports the following mark types: lines (*Line Tool*), rectangles (*Rectangle Tool*), ellipses (*Ellipse Tool*), text (*Text Tool*), and open or closed non-regular paths (*Pen Tool*). Similar as done in other design applications, the bounding box of the shape remains active after drawing for further manipulation. Data Illustrator relies heavily on paper.js [35] as the view model for rendering shapes in HTML5 Canvas.

Figure 7: Selecting shapes: (a) *Select Tool* selects entire shapes and collections of shapes. (b) *Direct Select Tool* selects anchor points and line segments of shapes.

Selecting Visual Components Selection is a prerequisite for operations such as changing visual properties, transforming objects, associating objects with data, or binding data to attributes. Data Illustrator supports two types of selection: (1) the *Select Tool* works on shapes and collections of shapes, (2) the *Direct Select Tool* works on anchor points and line/curve segments of shapes. The *Direct Select Tool* is a powerful feature in applications such as Illustrator, providing essential control to edit paths and deform regular shapes (e.g. rectangles). Both selection tools use familiar interactions such as: click to single select, shift+click to add to a selection, click+drag to lasso a selection, and clicking on the canvas to deselect. Selection tools are also used to transform objects: click+drag on an object to move it, click+drag on a bounding box corner to re-size a shape, or pressing an arrow key to nudge the selection. The rich selection of interactions in Data Illustrator provides the precise control required by designers. All user interactions in Data Illustrator are developed using RxJS [43] and Backbone.js [13].

Working with Data Data Illustrator allows users to work with one tabular dataset at a time. Users can choose from a spectrum of sample datasets from various sources, or upload a CSV file from their own computer. Upon loading a dataset, the system infers the data types of each column with the Datalib

library [19], displays data column summaries in the Data Variables panel, and shows the complete dataset in the Data Table panel. The Data Table also acts as an inspector for the data scopes of the currently selected visual items.

Context-Sensitive Interface We design the interface to be context-sensitive so that users can understand the possibility of actions at any state. The buttons in the Actionbar are enabled and disabled based on selection on the canvas. For example, if a group is selected, the "Partition" button is disabled, indicating that partitioning a group is not allowed. Similarly, the Property Inspector displays a set of property controls based on the shape type of current selection.

Repeating Repeat actions begin with the selection of a visual object (i.e. shape, group of shapes, or collection). Clicking the *Repeat* button displays a preview of how the selection will be repeated by a categorical variable. The preview supports changing the categorical variable. Upon confirmation, the repeat action duplicates the selected object, and places the two objects in a default grid layout. We chose to generate only two copies of the object because for large datasets, the number of objects will be overwhelming. To enable designers to control the number of objects to work on, we augment the *Repeat Grid* tool from Adobe XD (Figure8(a)). Users control the number of generated objects and the grid layout parameters by the following interactions: click+drag handles to display additional rows or columns, click+drag padding to adjust spacing, double-click to open the collection and select objects inside. Dragging past the total shapes allotted by data does not generate further repeated shapes.
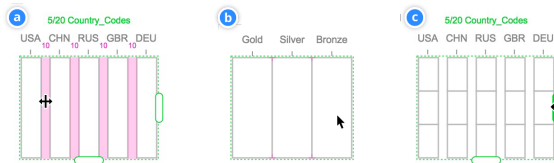

Figure 8: Collections with layouts: (a) Repeat Grid. (b) Partition Stack. (c) Partition Stacks Nested in a Repeat Grid

Partitioning Like the repeat action, the partition action requires a selection of a shape with or without a data scope. Groups or collections cannot be partitioned. Clicking the *Partition* button will display a preview of how the selected shape will be divided. Changing the data variable updates the preview auxiliary lines. Partitioning a rectangle results in a stack layout of the sliced rectangles. Unlike the grid layout, the stack layout does not support padding. Nested structures can be created by partitioning a shape multiple times - the data hierarchy limits the number of partition actions.

Peers Peers are the atomic objects created by repeat and partition actions. Similar to symbols in Sketch, or components in Figma, peers share visual properties. Upon selecting any shape, Data Illustrator highlights its peers with a faint blue to show linkage between peers. This design applies to anchor points and segments as well (Figure 9). Changes to a shape are instantaneously updated to its peer shapes. Properties shared between peers include: appearance, scaling, positioning
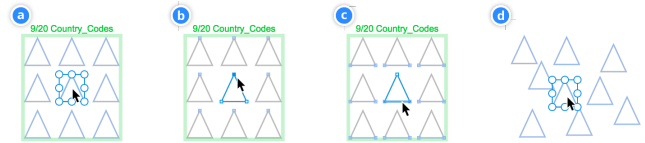

Figure 9: Peers highlighted on selection: (a) Peer shapes. (b) Peer anchor points. (c) Peer line segments. (d) Peer shapes in free-form layout.

anchor points or line segments, and data-bindings. The only non-linked property is shape position. Grid and stack layouts provide positioning to peers. When the user *breaks* a layout, the peer shapes can be positioned freely.

Breaking a layout is an irreversible action. Removing a grid layout poses a problem for controlling the number of displayed shapes without handles to reveal rows and columns. To remedy the loss of control over peer shape display, Data Illustrator provides a *Peer Count* slider in the Property Inspector.

Lazy Data-Bindings To map visual attributes to data, the user selects any object on the canvas. The Property Inspector populates with the corresponding set of properties. To bind data to property, tools such as Tableau or Lyra let users drag and drop a variable to a property field. We did not choose this design because dragging and dropping require significant cursor movement, and it is not clear which variables can be mapped to a given property. In our design, the user clicks the binding icon next to the property control, which displays a list of applicable data columns (Figure 10 left). Selecting a data column creates a data binding between the underlying data scope and that visual property for all peer objects. For each binding, the system creates a scale, where the range depends on the current values of the visual property. For example, position bindings use the bounds of all peer shapes, and continuous color bindings use the original hue of the selected object.
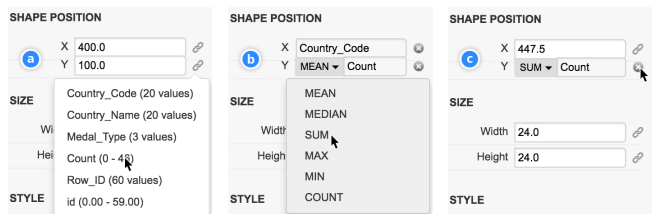
Figure 10: Lazy data-bindings: (a) clicking the binding icon shows a list of applicable variables, (b) changing the aggregator when binding numerical variables, (c) property control and icon update after binding, the remove icon indicates the possibility of removing a binding

Data-bindings are lazy in Data Illustrator, meaning that they constrain only their bound visual property. For example, position bindings only constrain the position of peer objects in relation to each other. Dragging a position-bound peer object will move the other peers and axis together. Lazy data-

bindings give designers control within a generative action. All data-bindings are implemented with the D^3 scale library [6].

Figure 11: Configurable axes and legends: (a) Numerical axis. (b) Categorical axis. (c) Categorical color legend. (d) Numerical color legend.
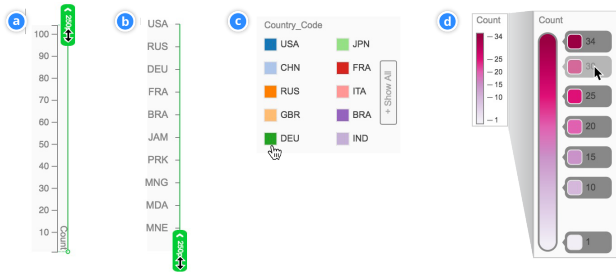
Interactive Axes and Legends Data Illustrator automatically creates interactive axes and legends upon successful completion of data bindings. Axes and legends in Data Illustrator are explanatory - acting as a reference for the data-binding, and configurable - supporting direct manipulation of the underlying scale. Upon their creation, Data Illustrator momentarily highlights axes and legends to help the user discover the afforded interactions. Users can adjust the offset of an axis with constraints: they can only move an x axis vertically, and a y axis horizontally. When the user moves an axis' peer objects, the axis follows, retaining its relative position. The user can click+drag on the axis handle to configure the underlying scale's range (Figure 11(a) and (b)), which updates the bound objects instantaneously. For categorical axis, users can define the order of objects in two ways: 1) sort the objects in a collection through the "ordered by" property control slider, 2) directly drag the axis text labels to reorder objects.

Color legends can be re-positioned anywhere on the canvas. They augment designs of color palettes and color gradients from mainstream vector editors. For categorical color legends, users change colors to replace the default colors picked by the system (Figure 11(c)). For numerical color legends, users can select the color for each stop, add a stop by a single click, and remove a stop by dragging it away (Figure 11(d)). Changes to a color legend are immediately applied to its bound peers. Furthermore, axes and legends of the same semantic type can be linked or merged to support consistent data-bindings across collections - a concept pioneered by Claessen and Wijk [9].

RELATED WORK

Visualization Grammars and Programming Toolkits
Graphical grammars [60, 59, 17] have served as the basis for powerful and expressive data visualization toolkits such as ggplot [58]. The grammars from Wilkinson [60] and Wickham [59] both follow a bottom-up approach: starting from data, aesthetic mappings from variables and coordinate systems drive the visual form of the graphic. Our framework assumes sketched shapes instead of data as the starting point. Data is only incorporated into graphics as necessary.

Declarative languages [17] provide a higher-level abstraction for constructing interactive visualizations by de-coupling specification from execution. Visualization toolkits built along this research direction [18, 5, 6, 29] simplify the construction of visualizations while preserving a broad design space. D^3 in particular, provides powerful capabilities in an accessible form, while supporting design freedom beyond data-driven mappings. Designers can create annotations, visual embellishments and additional structure alongside the declarative data mappings supported by D^3. The popularity of D^3 has been attributed to its re-use of an ubiquitous medium: the Document Object Model (DOM). Our approach strives to mimic this paradigm, as D^3 enables direct manipulation of the DOM, we augment vector editing tools to interact with the scene graph. While D^3 requires designers to select, bind, and style DOM elements with JavaScript code, we provide a visual and direct manipulation approach for non-programmers.

Interactive Visualization Design Tools
Numerous efforts support visualization authoring in a non-programming, interactive environment. Rost [46] reviews twelve of these interactive tools by comparing her experience re-creating Rosling's iconic GapMinder visualization [45] with each tool. Grammel et. al [16] provide a comprehensive survey of different categories of tools used to construct visualizations. Chart typology tools are a popular category of visualization design tools because of their ease-of-use. They provide users with a selection of templated charts to choose from (e.g. Many Eyes [55], DataWrapper [14], Raw [11], Plotly [42]). These tools quickly generate charts for users to compare design alternatives, however users are restricted to a predefined set of chart types and only a handful of customization options.

More robust and advanced tools such as Tableau provide approachable features such as recommending chart generation and automating best practices. Tableau is based on a table algebra framework [52], where operators such as cross and nest work solely on data. Visual marks only appear in a later stage of the pipeline. In our framework, operators such as repeat and partition primarily work on visual components, with data as ingredients. Tableau also supports detail-oriented customizations on scales and visual configurations, but these customizations often have to be accomplished through dialogs. Our approach brings the customization of scales, axes and legends directly to the canvas with direct manipulation widgets.

Recent data visualization research has sought to empower designers to create expressive visualizations without the need to program. Tools such as Lyra [48] and iVisDesigner [44] aim to provide users with the power of declarative toolkits in a familiar vector editing interface. Both systems employ higher-level representations of the scene graph: Lyra is built upon the Vega visualization grammar [29], while iVisDesigner's custom framework supports templated plots. User interaction modifies the abstraction, which in turn updates visualization rendering. In our approach, we do not have such abstractions: user interaction directly translates to operations on the scene graph. This choice allows us to focus on the interface and interaction design first, while the system architecture and visualization model come second.

Our work continues the line of research that explores visualization composition using graphical primitives. In early research such as SageBrush [47], users choose chart type from a list of prototypes, add graphemes to the prototype, and specify mappings between data and grapheme properties. SketchStory [33] and SketchInsight [34] use freeform shapes as archetypes to be repeated and transformed with data mappings. Victor [54] and Schachman [50] contribute procedural methods for designers to create parametrically constrained graphics. Professional design tools like Illustrator [23] and After Effects [21] provide features such as *Repeater* and *Blend Tool* to duplicate shapes and layers. These efforts inspire our work and suggest the need of a visual language that describes the composition and generation of diverse visualizations. Data-Driven Guides (DDG) [32] has taken a similar approach to ours by augmenting existing drawing tools and treating marks as flexible, deformable graphical elements. Our system supports a wider range of data-to-visual mappings and more complex layouts than DDG. Vuillemot and Boy [56] define a framework to assist designers in creating visualization mock-ups by employing top-down approach for subdividing the scene graph. Our framework is similar to the segmenting, nesting and linking portions of their framework. Our system creates high-fidelity visualizations instead of mock-ups.

Additional systems have investigated novel interaction techniques for visualization authoring. Early research explored programming by demonstration as a method for creating charts, and described heuristics for inferring user intention in chart specification [41]. iVoLVER [39] supports the extraction, transformation and presentation of information using pipeline style widgets in the canvas. Our system takes interaction design inspiration from the progressive disclosure techniques used in the iVoLVER system.

EVALUATION

Visualization Examples and Videos
We have created a diverse set of visualizations using the Data Illustrator system to demonstrate the expressivity of the framework. The visualization examples and videos showing the authoring processes are available at http://www.data-illustrator.com/gallery.php. In terms of chart type, the examples include rectangular bar chart, triangle bar chart, grouped bar chart, stacked bar chart, scatter plot, line graph, slope graph, bump chart, heatmap, parallel coordinates plot, range chart, Gantt chart, stringline chart, and small multiples.

User Study
Visualization design is a complex process involving multiple stages. In the initial phase, designers often generate ideas by sketching to explore the design space. When the visual form and structure are set, mockups are created to collect feedback [56]. In the final production stage, engineers or designers re-create high-fidelity visualizations by incorporating real data into the mockups. We focus on visualization re-creation tasks to evaluate whether designers can understand and use the framework to compose visualizations. While the re-creation task is not an exact replica of the complete design process, it allows us to choose visualizations that cover all the concepts and features in our tool, and to compare participants' performance objectively. Furthermore, the ability to think and act in terms of the framework concepts is the cornerstone of using Data Illustrator for ideation and more open-ended designs.

We recruited 13 designers (7 male, 6 female) from the Puget Sound area and the Atlanta metropolitan area. The breakdown of their experiences in graphic design is as follows: less than 2 years: 1 (7%); 2-4 years: 3 (23%); 4-6 years: 6 (46%); 6-8 years: 1 (7%); greater than 8 years: 2 (15%). Their design work included web UI (85%), mobile UI (77%), visualization and infographics (61%), graphics and illustration (54%), print (54%), logo (38%) and video game (7%). Out of the 13 participants, 5 (39%) had minimal or less than 2 years of experience with visualization, 4 (31%) had 2-4, 2 (15%) had 4-6, and 2 (15%) had more than 6 years of experience.

The study with each participant lasted 1.5 hours. In the setup we used two monitors, each with a resolution of 2500x1600. We first gave a tutorial on Data Illustrator following the script at https://goo.gl/UtZruK. The participants learned about the main features of the system by creating three simple visualizations: a stacked bar chart, a scatter plot, and a triangle bar chart with both positive and negative values. The tutorial lasted around 35 to 40 minutes. The participants then were asked to complete two visualization creation tasks. At the end of the two tasks, they could decide if they wanted to work on an optional, more difficult task, if time allowed. We used "Obesity vs. Education" for Task 1, and "NBA Redraft" for Task 2. The optional Task 3 was to create the "Red and Blue America" visualization. These three visualizations covered the main features and functionality of the system: Task 1 requires repeating a line by data, binding categorical data to the position and colors of anchor points, and merging scales; Task 2 involves both repeat and partition, binding numerical data to fill color, and nested collections; Task 3 requires nested collections, binding data to segment position and fill color, and breaking layout for position binding. For each visualization, we explained the schema and meaning of the source data, and described what the visual components and their properties represent. We asked the participants to focus on the main visualization and not to worry about the annotations. At the end of the session, each participant completed a questionnaire and answered questions in a semi-structured interview.

All participants successfully completed Tasks 1 and 2 with minimal guidance (μ=12.23 minutes, σ=5.61 for Task 1, μ=10.77 minutes, σ=4.30 for Task 2). Out of 13 participants, 12 volunteered to work on the third task. Four of them completed it successfully (μ=14.75 minutes, σ=2.87), the remaining eight could not finish the task after spending time (μ=10.63 minutes, σ=5.93). For these eight participants, we analyzed how close they were from success. Completing Task 3 required three milestone steps to be accomplished: create nested collections with Year and State, position the states on a map layout, and bind PVI to the y position of the rectangles' top segments. Seven out of the eight participants were able to finish two of the steps but were stuck on the last step.

The participants rated their experience of learning and using Data Illustrator on a 5-point Likert scale. The results are as

follows: on learning, μ=2.62, σ=0.96 (1-very easy, 5-very difficult); on creating visualizations, μ=2.38, σ=0.77 (1-very easy, 5-very difficult); on the authoring experience, μ=2.15, σ=0.90 (1-very enjoyable, 5-very frustrating).

Designers' background and expertise directly affected their learning experience and performance. For those who had substantial experience with visualization, they thought learning was easy: "*Tableau has a bit of a learning curve, and with Data Illustrator being based off of Adobe Illustrator, there isn't as much of a learning curve.*" (P3). In contrast, P5 had little experience with visualization, and compared to learning with graphic design tools: "*it takes 30 minutes for me to learn the [Data Illustrator] tutorial via a person, that usually to me is not an easy program. [Adobe] XD for me was easy 'cause I didn't have to use any tutorials, so I'd say [learning with Data Illustrator] is somewhat difficult*".

The participants were impressed by the power of the tool: "*Very impressive. When I looked at all 3 visualizations I thought: oh boy, how am I going to do this! Then once you finally work through the sequences needed to make it, the actually-doing-it part is super easy!*" (P11). P9 commented on the tool's flexibility and ease of use: "*I feel like it's more flexible than D^3 or Tableau. It's a happy medium of being able to control the graphic visually. It's pretty simple too, you don't have to be a super expert user like with Adobe Illustrator, which is nice. It's a nice sweet spot between having little control with Tableau and getting frustrated with D^3*". P6, however, knew little about visualization and did not understand the concept of scale. He struggled in the authoring process, but still managed to complete the two tasks by trial and error.

We also observed that the designers exhibited different workflows in the authoring processes. In Task 1, some participants used the *Repeat Grid* to generate a few lines first, bound data to the anchor point positions, then used the *Peer Count* slider to generate the remaining lines; other participants generated all the lines first before binding data to positions. The strategies to accomplish Task 2 also varied. Some participants generated all the rectangles by Row ID, broke the grid, and bound data to the x and y positions; others saw a nested structure in the visualization, and repeated a rectangle by Year then partitioned the rectangles by Player. This diversity of workflow demonstrates the flexibility of our framework and system.

We identified three recurring pain points in using the system. First, many participants confused the order of shapes inside a collection with their positions. In Task 2, they wanted to generate the visualization by simply sorting the shapes in a *Repeat Grid*. The order did determine the shapes' positions in a collection to a certain extent, which was the source of confusion. Second, in the current design, to bind data to shapes' position, one must break the layout. Otherwise the position property controls are hidden. Some participants were baffled not seeing the position controls. Showing the position controls at all times and prompting to break layout can resolve this problem. Finally, several participants could not recall the feature of binding to segment position and adjusting the scale to generate bar charts with negative values in Task 3. Binding

to height felt more natural to them. We plan to handle negative values when binding data to height to address this issue.

The participants also made suggestions on how to improve the interface and system. The lack of undo functionality bothered many participants. They were afraid of making irreversible mistakes and chose to think deeply about the authoring strategy before trying it out. They also wanted a more robust *Pen Tool* and the ability to style the axes and draw grid lines. Some participants also commented that starting from scratch was harder than picking a template: "*It took me some time [...] to think through sequences I would need to take to re-create it. Tableau has the 'Show Me' feature that hints what bindings can be made with that type of dataset*" (P3). Such comments are consistent with previous research findings [38]: compared to automated visualization generation tools, design-centric authoring tools evoke deeper reflections on design choices and execution plans. For use cases where quick visualization construction is desirable, saving the visualizations as reusable templates will be very useful.

CONCLUSION AND FUTURE WORK

We present a novel framework that describes the generation of visualizations from the perspective of graphic design. Based on the framework, we design and build Data Illustrator, a system that augments vector design tools with lazy data binding for visualization authoring. We demonstrate the expressivity of our approach through visualization examples. A study with 13 designers shows that the system is learnable and designers can use the framework to compose visualizations. The Data Illustrator system is available at `http://www.data-illustrator.com`.

The framework we created provides descriptive and generative power for visualization design, but in its current form, it is not complete. The framework needs to be expanded to include area as a shape primitive, so that it can describe visualizations such as area charts and stream graphs. Further research is also necessary to include the support for hierarchical, network and geographic data, and the corresponding visualizations.

In terms of implementation, Data Illustrator demonstrates the feasibility and power of a subset of the framework. Layouts such as packing and polar coordinate positioning are not yet supported in the system. We also need to implement ring as a shape primitive for creating donut charts. Finally, systems such as Lyra build on top of Vega, which have access to the functionalities offered by D^3, including interpolation methods for lines and curves. Data Illustrator should provide those capabilities in order to be more powerful.

For future work, we would also like to explore how to turn visualization designs into reusable templates. Once users create a visualization inside Data Illustrator, they should be able to export it into formats readable by other tools and to share it with other users, who can customize the design with their own data and visual styles. Adding authoring support for interactivity is also a direction to investigate further.

REFERENCES

1. Accurat. 2014. Brain Drain - part of the collection: La Leturra - Visual Data. (2014). Retrieved September 17, 2017 from `https://www.accurat.it/works/la-lettura/`.

2. Anima App. 2017. Auto-Layout: Introducing Stacks-Flexbox for Sketch. (21 February 2017). Retrieved September 17, 2017 from `https://medium.com/sketch-app-sources/auto-layout-introducing-stacks-flexbox-for-sketch-c8a11422c7b5`.

3. Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. 2014. Reflections on how designers design with data. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*. ACM, 17–24. DOI: `http://dx.doi.org/10.1145/2598153.2598175`

4. Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. 2017. Iterating between tools to create and edit visualizations. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (January 2017), 481–490. DOI:`http://dx.doi.org/10.1109/TVCG.2016.2598609`

5. Michael Bostock and Jeffrey Heer. 2009. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (December 2009), 1121–1128. DOI: `http://dx.doi.org/10.1109/TVCG.2009.174`

6. Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3: data-driven documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (January 2011), 2301–2309. DOI: `http://dx.doi.org/10.1109/TVCG.2011.185`

7. Alberto Cairo. 2012. *The Functional Art: An introduction to information graphics and visualization*. New Riders.

8. Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. 1999. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.

9. J. H. T. Claessen and J. J. van Wijk. 2011. Flexible Linked Axes for Multivariate Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (December 2011), 2310–2316. DOI: `http://dx.doi.org/10.1109/TVCG.2011.201`

10. Bohemian Coding. 2017. Sketch - The digital design toolkit. (2017). Retrieved September 17, 2017 from `https://www.sketchapp.com/`.

11. Density Design and Calibro. 2013. Raw Graphs. (16 January 2013). Retrieved September 17, 2017 from `http://rawgraphs.io/`.

12. Figma Design. 2017. Figma: the collaborative interface design tool. (2017). Retrieved September 17, 2017 from `https://www.figma.com/`.

13. DocumentCloud. 2016. Backbone.js. (5 April 2016). Retrieved September 17, 2017 from `http://backbonejs.org/`.

14. Datawrapper GmbH. 2017. Datawrapper. (2017). Retrieved September 17, 2017 from `https://www.datawrapper.de/`.

15. Russell Goldenberg. 2017. Twenty Years of the NBA Redrafted. (2017). Retrieved September 17, 2017 from `https://pudding.cool/2017/03/redraft/`.

16. Lars Grammel, Chris Bennett, Melanie Tory, and Margaret-Anne Storey. 2013. A survey of visualization construction user interfaces. *EuroVis - Short Papers* (2013), 19–23.

17. Jeffrey Heer and Michael Bostock. 2010. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (December 2010), 1149–1156. DOI: `http://dx.doi.org/10.1109/TVCG.2010.144`

18. Jeffrey Heer, Stuart Card, and James Landay. 2005. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, 421–430. DOI:`http://dx.doi.org/10.1145/1054972.1055031`

19. Jeffrey Heer, Dominik Moritz, Michael Correll, and Kanit Wongsuphasawat. 2017. Vega Datalib. (26 October 2017). Retrieved September 17, 2017 from `https://github.com/vega/datalib`.

20. Nigel Holmes. 1984. *Designer's guide to creating charts & diagrams*. Watson-Guptill.

21. Adobe Systems Inc. 2017a. Adobe After Effects CC | Visual effects and motion graphics software. (2017). Retrieved September 17, 2017 from `http://www.adobe.com/products/aftereffects.html`.

22. Adobe Systems Inc. 2017b. Adobe Experience Design CC (Beta) | Prototyping & Wireframing Tool. (2017). Retrieved September 17, 2017 from `http://www.adobe.com/products/xd.html`.

23. Adobe Systems Inc. 2017c. Adobe Illustrator CC | Vector Graphic Design Software. (2017). Retrieved September 17, 2017 from `http://www.adobe.com/products/illustrator.html`.

24. Adobe Systems Inc. 2017d. Adobe InDesign CC | Desktop Publishing Software and Online Publisher. (2017). Retrieved September 17, 2017 from `http://www.adobe.com/products/indesign.html`.

25. Adobe Systems Inc. 2017e. Adobe Photoshop CC | Best photo, image & design editing software. (2017). Retrieved September 17, 2017 from `http://www.adobe.com/products/photoshop.html`.

26. Adobe Systems Inc. 2017f. Cut, divide, and trim objects in Illustrator. (10 March 2017). Retrieved September 17, 2017 from `https://helpx.adobe.com/illustrator/using/cutting-dividing-objects.html`.

27. Adobe Systems Inc. 2017g. Draw circular (Polar) grids. (13 September 2017). Retrieved September 17, 2017 from `https://helpx.adobe.com/illustrator/using/drawing-simple-lines-shapes.html`.

28. Adobe Systems Inc. 2017h. How to blend objects in Illustrator. (15 February 2017). Retrieved September 17, 2017 from `https://helpx.adobe.com/illustrator/using/blending-objects.html`.

29. University of Washington Interactive Data Lab. 2017. Vega - A Visualization Grammar. (25 October 2017). Retrieved September 17, 2017 from `https://vega.github.io/vega/`.

30. Waqas Javed and Niklas Elmqvist. 2012. Exploring the design space of composite visualization. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*. IEEE, 1–8. DOI: `http://dx.doi.org/10.1109/PacificVis.2012.6183556`

31. Kantar. 2017. Information is Beautiful Awards. (2017). Retrieved September 17, 2017 from `https://www.informationisbeautifulawards.com/`.

32. Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilmot Li, Jovan Popovic, and Hanspeter Pfister. 2017. Data-driven guides: Supporting expressive design for information graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (January 2017), 491–500. DOI: `http://dx.doi.org/10.1109/TVCG.2016.2598620`

33. Bongshin Lee, Rubaiat Habib Kazi, and Greg Smith. 2013. SketchStory: Telling More Engaging Stories with Data Through Freeform Sketching. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (December 2013), 2416–2425. DOI: `http://dx.doi.org/10.1109/TVCG.2013.191`

34. Bongshin Lee, Greg Smith, Nathalie Henry Riche, Amy Karlson, and Sheelagh Carpendale. 2015. SketchInsight: Natural data exploration on interactive whiteboards leveraging pen and touch interaction. In *Visualization Symposium (PacificVis), 2015 IEEE Pacific*. IEEE, 199–206. DOI: `http://dx.doi.org/10.1109/PACIFICVIS.2015.7156378`

35. Juerg Lehni and Jonathan Puckey. 2015. Paper.js. (25 October 2015). Retrieved September 17, 2017 from `http://paperjs.org/`.

36. Malcom Maclean. 2014. d3.js multi-line graph with automatic (interactive) legend. (8 July 2014). Retrieved September 17, 2017 from `https://bl.ocks.org/d3noob/08af723fe615c08f9536f656b55755b4`.

37. Philip B Meggs and Alston W Purvis. 2016. *Meggs' History of Graphic Design*. John Wiley & Sons.

38. Gonzalo Gabriel Méndez, Uta Hinrichs, and Miguel A. Nacenta. 2017. Bottom-up vs. Top-down: Trade-offs in Efficiency, Understanding, Freedom and Creativity with InfoVis Tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, 841–852. DOI: `http://dx.doi.org/10.1145/3025453.3025942`

39. Gonzalo Gabriel Méndez, Miguel Nacenta, and Sebastien Vandenheste. 2016. iVoLVER: Interactive Visual Language for Visualization Extraction and Reconstruction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, 4073–4085. DOI: `http://dx.doi.org/10.1145/2858036.2858435`

40. Josef Muller-Brockmann. 1985. *Grid systems in graphic design: A visual communication manual for graphic designers, typographers and three dimensional designers*. Arthur Niggli.

41. Brad Myers, Jade Goldstein, and Matthew Goldberg. 1994. Creating charts by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. ACM, 106–111. DOI: `http://dx.doi.org/10.1145/191666.191715`

42. Plotly. 2017. Plotly. (2017). Retrieved September 17, 2017 from `https://plot.ly/`.

43. ReactiveX. 2016. RxJS. (12 December 2016). Retrieved September 17, 2017 from `http://reactivex.io/rxjs/`.

44. Donghao Ren, Tobias Hollerer, and Xiaoru Yuan. 2014. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (December 2014), 2092–2101. DOI: `http://dx.doi.org/10.1109/TVCG.2014.2346291`

45. Hans Rosling. 2015. Gapminder. (2015). Retrieved September 17, 2017 from `https://www.gapminder.org/tools/`.

46. Lisa Charlotte Rost. 2016. One Chart, Twelve Tools. (17 May 2016). Retrieved September 17, 2017 from `http://lisacharlotterost.github.io/2016/05/17/one-chart-tools/`.

47. Steven Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. 1994. Interactive Graphic Design Using Automatic Presentation Knowledge. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. ACM, 112–117. DOI: `http://dx.doi.org/10.1145/191666.191719`

48. Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum* 33, 3 (2014), 351–360. DOI: `http://dx.doi.org/10.1111/cgf.12391`

49. Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (January 2017), 341–350. DOI: `http://dx.doi.org/10.1109/TVCG.2016.2599030`

50. Toby Schachman. 2015. Apparatus: A hybrid graphics editor and programming environment for creating interactive diagrams. (10 November 2015). Retrieved September 17, 2017 from `http://aprt.us/`.

51. Tableau Software. 2017. Tableau Software: Business Intelligence and Analytics. (2017). Retrieved September 17, 2017 from `https://www.tableau.com/`.

52. Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (December 2002), 52–65. DOI:`http://dx.doi.org/10.1109/2945.981851`

53. John Thompson and John Stasko. 2016. Understanding Data-Driven Visual Encodings through Deconstruction. *Poster at IEEE VIS 2016* (2016). `https://www.cc.gatech.edu/~stasko/papers/infovis16-poster-deconstruction.pdf`

54. Bret Victor. 2013. Drawing Dynamic Visualizations. (2013). Retrieved August 3, 2016 from `https://vimeo.com/66085662`.

55. Fernanda B Viegas, Martin Wattenberg, Frank Van Ham, Jesse Kriss, and Matt McKeon. 2007. Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (December 2007), 1121–1128. DOI: `http://dx.doi.org/10.1109/TVCG.2007.70577`

56. Romain Vuillemot and Jeremy Boy. 2018. Structuring Visualization Mock-ups at the Graphical Level by Dividing the Display Space. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (January 2018), 424–434. DOI: `http://dx.doi.org/10.1109/TVCG.2017.2743998`

57. Jagoda Walny, Samuel Huron, and Sheelagh Carpendale. 2015. An Exploratory Study of Data Sketching for Visual Representation. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 231–240. DOI: `http://dx.doi.org/10.1111/cgf.12635`

58. Hadley Wickham. 2009. *ggplot2: elegant graphics for data analysis*. Springer Science & Business Media.

59. Hadley Wickham. 2010. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28. DOI: `http://dx.doi.org/10.1198/jcgs.2009.07098`

60. Leland Wilkinson. 2005. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc.

61. Randy Yeip, Stuart A Thompson, and Will Welch. 2016. A Field Guide to Red and Blue America. (25 July 2016). Retrieved September 17, 2017 from `http://graphics.wsj.com/elections/2016/field-guide-red-blue-america/`.