A View-level Abstraction for Coordinating D3 Visualizations

Omkar S. Chekuri*
School of Computer Science
The University of Oklahoma

Chris Weaver†
School of Computer Science
The University of Oklahoma

ABSTRACT

D3 is a popular and effective library for the development and deployment of visualizations in web pages [6]. Numerous applications testify to its accessibility and expressiveness for representing and manipulating page content. By eschewing toolkit-specific abstraction, D3 gains representational transparency, but at a cost of modularity. Few D3 applications compose multiple views or support coordinated interaction beyond simple brushing. Rather than relegate view composition to custom integration code, we overlay D3 with a view-level abstraction that utilizes a general parameter sharing model to offer simple yet flexible composition of coordinated multiple views (CMV) while preserving the expressiveness of individual D3 components. Coupling of event handling to shared parameters recasts modeling of interactive state and simplifies declarative specification of interactive dependencies between views. We present two examples of extensively coordinated visualizations and demonstrate how view-level abstraction can radically reduce code to compose complex D3 visualizations.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools—Visualization toolkits

1 INTRODUCTION

Data-Driven Documents (D3) [6] has become one of the most widely used libraries for embedding visualizations into web pages. It offers an accessible, expressive, declarative language for representing and transforming data accessed via a document object model (DOM) of a page. It also supports manipulation of data representation by linking interaction events to transformations. These capabilities have facilitated development of a panoply of examples. The world of D3 visualizations leans heavily toward simpler designs appropriate for communication and explanation. The relative rarity of composite designs, like the CMV constructions commonly used in exploration and analysis applications, nevertheless seem surprising.

It is less surprising when one considers D3's goal to avoid a toolkit-specific abstraction of visual representation. Unlike in many visualization systems, "view" is not a explicit concept in D3. What constitutes a view is largely up to the developer to determine. D3 can be readily used to transform page structure and styling in ways that don't corresponding to commonly recognized view types. Moreover, D3 exerts relatively heavy influence over interaction handling in its representations. Its encapsulation of event processing makes it difficult to combine channels of interaction if their events conflict, for instance if dragging triggers both brushing and panning [17]. Composite visualizations commonly contain not only multiple views, but also multiple channels of interaction in many of those views, to support the mix of navigation, selection, and other manipulation operations the user may want to perform [27]. The more views there are, and the more complex the interactions within and between those views, the more complicated it becomes to implement the corresponding dependencies between the D3 fragments that correspond to each "view" (and its corresponding portion of the DOM). While neither the specification of layouts in D3's descendant Vega-lite [17] nor the reactive semantics of visualization specification added in Reactive Vega [18] directly addresses general CMV composability in visualization design, both offer complementary capabilities.

Here we present work-in-progress on developing a view-level abstraction for coordinating multiple views in a single D3 visualization. Models of coordination, and how they abstract interaction state and the interaction dependencies between views, have long been of interest particularly in information visualization. Constraint-based user interfaces like Garnet [13] and Rendezvous [11]), and seminal visualization systems such as Tioga-2 [4], Visage [16], Spotfire [1], and XGobi [8], heavily influenced work on CMV systems including DEVise [12], Snap-Together Visualization [15], GeoVISTA *Studio* [21], CViews [7], IVEE [3], and Improvise [22]. Our experiences and others' in building Improvise visualizations makes us keenly aware of the persistent gulf between accessibility and expressiveness for visualization designers, and how difficult it can be to bridge the capabilities of visualization systems [10].

With these challenges in mind, we set out to adapt the Live Properties architecture [22] to serve as the core of a new CMV abstraction layer implemented in JavaScript on top of D3. In pursuing this goal we have six specific objectives: **transparency**, preserving accessibility and expressiveness of D3 code in view implementations; **extensibility**, supporting open-ended creation and variation of new view types; **modularity**, allowing any number of view instances of any type; **composability**, allowing coordination between arbitrary sets of view instances as independent objects; **consistency**, producing identical appearance and behavior across local or coordinated updates, within performance limits; and **consonance**, supporting harmonious mix-and-match of multiple coordinations between views.

In this paper we present the following contributions: an implementation of the Live Properties architecture in JavaScript; the design and implementation of a common wrapper for D3 views that interface with Live Properties; a library of several common view types, each implemented as D3 code inside the wrapper; two examples of composite visualizations with extensive navigation and selection coordination to demonstrate the design scope of the package; and an assessment of our progress in satisfying the objectives above. Our results confirm that the coordination abstraction supports rapid, incremental combination of a readily expandable gallery of D3 components into flexible navigation and selection coordination patterns.

2 VIEW ABSTRACTION

View Abstraction is achieved by providing a wrapper for the D3 views. D3 components are fragmented into three sections inside the view modules. Firstly, there is a constructor that is responsible for initialising the instance variables of the view.Secondly, there is a "Render" component this is responsible for creating the required DOM structure and create the view drawn using the default parameters and visual encoding. Thirdly, there are various update methods that are responsible for modifying the view when ever a property change defining a particular behaviour or appearance occurs. Special adjustments to D3 are made in the render method to facilitate coordination and multi-view layout.Views have the capability to conditionally attach the axis and labels to the view based on the user defined parameters during view initialization. Each views are

*e-mail: omkar.chekuri@ou.edu
†e-mail: cweaver@ou.edu

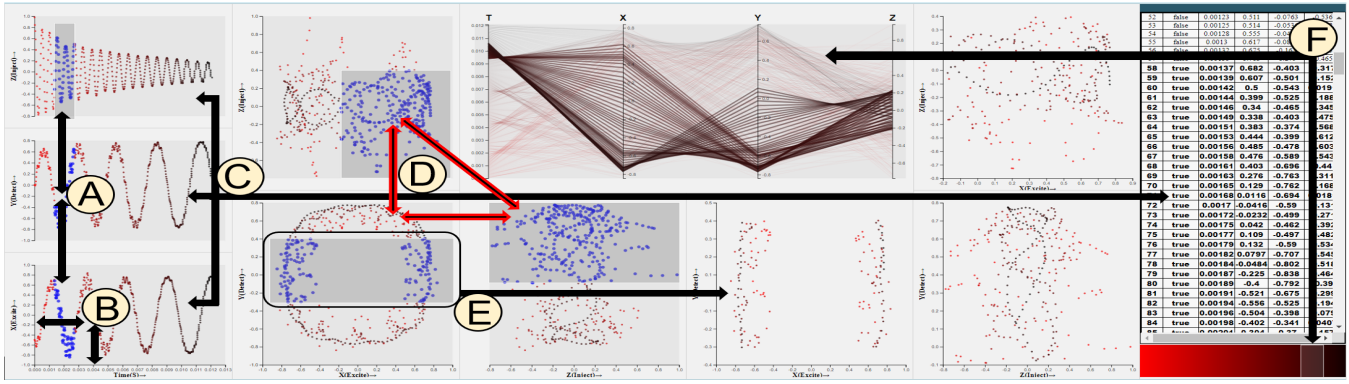

Figure 1: Reproduction of the cubic ion trap visualization [22] using 12 coordinated D3 views. (A) Three time series synchronize scrolling horizontally. (B) Independent scrolling. (C) Shared selection between time series and table. (D) Overview SPLOM showing three sides of the cube. (E) Detail SPLOM zoomed to ranges selected in the overview. (F) Range selected in a color slider filters data in a parallel coordinate plot.

attached to a HTMLDIVELEMENT element that have the option to let the user drag each view in a visualization if the user intends to organize the layout on the fly.These HTMLDIVELEMENT's are used for positioning the view in default layout. The views were designed to share a properties that they all can interpret and use e.g. we used a Boolean array for selection to highlight the list of selected items in any view. All the views share the same data that have a same index number assigned for each row. Whenever selection or indication is done on the view the view is updated based on the bit array whose length is same as the data attached to the view and the index number of the bits in the bit array will match with the index number of the attached data. The views use a property of type NUMBER to perform coordinated navigation , it uses a property of Array type to perform shared selection and shared navigation by using a bit array.

There are four cases when a view is drawn (Fig. 2). Firstly, instantiating the control will draw the view using the default parameters and visual encoding which is represented as INITIAL VISUAL ENCODING . Secondly, when the user assigns default values to the properties of the view wrapper when he binds the variable to the control, the view is updated to reflect the default values of the properties using appropriate methods which is represented as LIVE PROPERTY VISUAL ENCODING , Thirdly, whenever a user performs an interaction, If the interaction has any decoration associated with such as grey rectangle in case of brushing the decoration is drawn which is represented as INTERACTION VISUAL ENCODING . Fourthly, any interaction in the view by the user will create an event which that the user uses to create a value for the live property. e.g. The user uses the brush event to select the containment of points in the region and creates bit array to represent the state of the data item. This value will trigger a series of events and is sent to all the views that are coordinated. The views use this value to draw the visual encoding the user chooses for the particular interaction. Writing the methods to make changes to the view

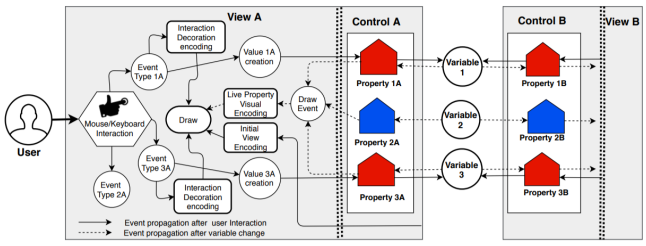

Figure 2: The data flow graph representing the event propagation, view creation, view redraw and coordination mechanism between views drawn using D3 and Live Properties Architecture.

Most often selection and indication interaction are not separated clearly in coordinated visualizations.From a designers point of view, selection and indication are very similar as they both are implemented and coordinated between different visualizations by a similar manner using but from users point of view both are Completely different. Indication is a behaviour that is used for changing items but its not the thing that is changing. The semantics of how these are specific to the view implementation and not specific to live properties. By differentiating indication from selection using two separate Live Properties for the visualization, we were able to use the brush in two different kind of modes, transient and lasting [5]. We used indication property for the transient mode and the Selection Live Property for Lasting mode . For Example When we made a selection in the Table we used as a lasting mode as hovering over the table does not effect the selected items in the map, where as whenever we brush in Map view in transient mode and the Hovering area changes on the map, the indicated item on the scatter plot gets highlighted and changes.

Live properties is a user interface architecture for directly coordinating controls (Widgets, D3-views) through shared objects [22]. They serve two main purposes. They act as value slots for determining the appearance and behaviour of views. They also act as ports through which controls communicate as a result of interaction. The views are designed using live properties as instances of model view controller architecture with many small models. Each view has a control and each control has many properties associated with it that determine how the view appears and behaves (Fig. 2). Live properties lets us describe how views coordinate in a centralized manner. This greatly simplifies the D3 visualizations and improves the performance of the D3 visualizations by avoiding cycles that are detrimental to the performance of visualizations in browser based environment. All the properties can be used for initializing the views with their default values, but only some are responsible for conveying the value change triggered by the interaction of the view to the variable that they are bound to. These are called Active properties. The properties that are used only used for modyfying the views are called passive properties. Property 2A and 2B in (Fig. 2) represent "Passive Properties", Property 1A, 3A, 1B and 3B represent "Active properties". The different software and development environment used to develop this architecture are a windows 10 Machine,Chrome browser, Visual Studio code editor. All the views were drawn using D3 version 5.16.o .

In the example shown in (Fig. 2), we first Initialize Global variables here (Variable 1, Variable 2 and Variable 3) . These variables have types that represent the values that these variables can store such as Array, Integer, String, Double etc.These variables also have default values. We then initialize Controls here (Control A, Control B) . These controls initialize the Properties shown in pentagons

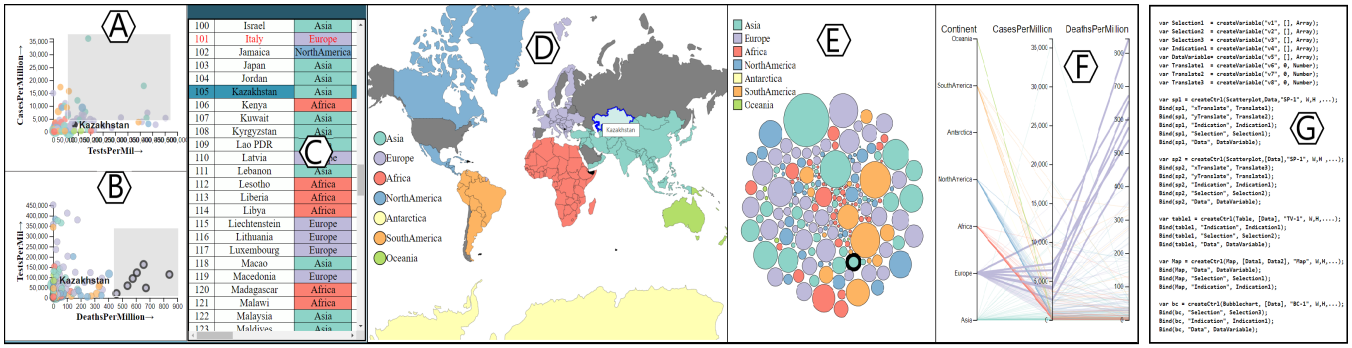

Figure 3: Visualization of COVID-19 data [26] with six views (A–F): two scatterplots, a table, a choropleth world map, a bubble chart, and a parallel coordinate plot. Views encodes different attributes of countries using a common color scheme for continents. Lightly abridged code to generate the layout and coordinations is at right (G). Coordination between views includes range navigation (AB), selection (AD, BCF), and indication (ABCDE).

corresponding to them and creates their respective views with default visual encoding . The properties associated with Controls has types associated with them that represent the type of value these can store such as Array, Integer, String, Double etc. We then bind the Properties of Controls with the variables whose types match. As soon as we bind the variables with the properties the default values of the variables are propagated to the properties. These properties then update the visual encoding of the views associated with them. Whenever and interaction is performed in the views such a rubber band, an event is generated that will update the visual encoding of the interaction such as grey colored rectangle in case or rubber band selection or title in case of hover. The event that is triggered will populate the value of the property variable locally and passes this value to the variable bound to the relevant live property. The variable then updates the values of all the live properties that are bind to it. The updated live properties then trigger a draw event that is propagated to the view. The view now gets the value of the live property through this event. The views then decide how to change the visual encoding using this value.The views discard some events if we do not define a call back function to perform some action.

The core functionality of the live properties Architecture [22] is re-implemented in JavaScript using typescript which was originally in Java. Designers need to have a basic understanding of how to construct D3 views in typescript to make the architecture work together. Issues may arise with the use of ”this” keyword as it may refer to the instance of the class or it may refer to the current selection in D3. Whenever the designer intends to use instance variables of the class always make a local copy of them inside the method and use this local local copy as it avoids unintended results. We used some of the most common data types such as arrays, Boolean values, numbers, strings and functions as values for the properties.

3 Examples of Coordination

With multiple views in a visualization, users can see and interact with data from multiple perspectives simultaneously. Coordination allows users to combine the interactions of different views to compose more complex queries than would be possible in any single view. To support common data exploration strategies [19], many visualizations employ basic navigation and selection coordinations [14], such as brushing [5]. Techniques like compound brushing [9] and cross-filtering [20, 25] employ coordination constructions that interplay with data processing and querying in more complex ways.

While our current view abstraction focuses on supporting basic forms of coordination, it is sufficient to realize many common coordination patterns [24], and visualization designers can vary and synthesize them to suit specific application needs. Transparency of D3 code in views also leaves the door open for designers to implement direct view dependencies if they so desire. We demonstrate both through two example visualizations. Figure 1 shows a reproduction of the original Improvise visualization of 3D ion motion in a magnetic field [22], exemplifying several navigation coordination patterns. In figure 3, six different views of COVID-19 data [26] are heterogeneously coordinated via a mix of several selection, indication, and range parameters. View instantiation, layout, and coordination can be varied incrementally and immediately with very little code (fig. 3G). Adding each view type (figure 3A–F) required very little effort beyond writing its particular D3 code. Each example demonstrates how multiple different coordination patterns can be easily and effectively combined in visualization designs.

In fig. 1A, *synchronized scrolling* coordinates the horizontal range of time visible in three time series plots. Each plot has a pair of live properties to represent its horizontal and vertical visible ranges, each corresponding to a translated axis value in D3. Changes to either range are applied to the corresponding axis and the points in the plot are re-positioned accordingly. Horizontal axis labeling was turned off in the upper two plots to remove the redundancy and create a simpler, more compact, more appealing layout. Splitting horizontal and vertical ranges into two separate live properties offers flexibility in how coordinate systems can be interactively linked across views, with the added benefit of increasing freedom of multiview layout. Synchronized scrolling is varied in this way in fig. 3A–B to coordinate the horizontal axis of the upper scatterplot with the vertical axis of the lower scatterplot. Flexible binding of variables to view properties enables navigational coordination of views even when they display different data attributes or use different visual mappings to determine position.

A more complex construction of range bindings yields the *scatterplot matrix* shown in fig. 1D. The SPLOM of three data dimensions is laid out in the traditional stair step arrangement of three plots. X, Y, and Z range variables are bound to the range properties of each plot so as to create the expected synchronization of scrolling across all three views. The trick is to use Z for the horizontal of the ZY plot but the vertical of the XZ plot, which is straightforward using live properties. Two such SPLOMs can be coordinated to create the navigational *overview + detail* pair shown in fig. 1E; in each of the overview plots, a D3 brush (gray rectangle) delimits a 2D region of space that matches the full extent of the corresponding detail plot. The coordination itself happens by exposing the horizontal and vertical extents of the brush rectangle as range properties, bound to the same range variables as the detail plot. An alternative way to implement the overview + detail coordination would share the selection of data inside the brush bounds instead of the bounds themselves. The detail plot would share a selection variable with the overview plot, but use that selection to calculate its extent as a bounding rectangle that contains only the selected data (plus padding). This approach would still work if the overview and detail plots used different visual encodings, with the coordination more clearly a selection than a navigation. The view abstraction allows tailoring of D3 code to offer

standard or custom coordination semantics.

Shared selection is used in fig. 1 to support brushing between the three time series plots and the table view. The three plots in the SPLOM are similarly coordinated, but via a second selection variable. Separation of brushed views into two groups lets the user brush the data relative to time or 3D space independently. The visualization in fig. 3 similarly coordinates hover interactions in most of its views to allow *shared indication*, in this case of a single country. Much like in shared selection, data items hovered over in any of the coordinated views is somehow highlighted in all of those views. How those items are highlighted is determined in each view using the indication property value to filter or encode indicated items differently from other items, e.g. as a label in the map, a thick outline in the bubble chart, and a teal row background in the table. All the views have both selection and indication properties, allowing flexible mix-and-match of hovering and brushing highlighting.

The color slider and parallel coordinate plot in fig. 1F provide an example of a custom dependency between views. The two views share a range variable for the selected range of color in the color gradient. The dependency arises because the two views also need to share a color gradient for sake of visual consistency. In this case we implemented both as general-purpose views but tweaked their respective D3 code to use the same gradient by coincidence. Adding a color gradient property to both views is an alternative approach. Both approaches warrant consideration of how customization risks proliferation of view types. Support for coordination beyond navigation and selection, providing interactive access to the full visualization data processing pipeline, is a clear direction for future development of the view abstraction.

4 Discussion

Our foremost objective was to preserve **transparency** of D3 code in view implementations. By parameterizing views via minimal sets of properties needed to share coordination state, we were able to preserve the accessibility. By giving the view responsibility over how to use property values to determine appearance and behaviour, we were able to preserve expressiveness. The visualization designer can choose the visual encoding appropriate to the data and application, and can also implement interaction as needed to accomplish a given task. Adjusting the selection in a view can involve click, rubber band, lasso, or other selection gestures. Indication by drawing or hovering is similarly flexible. In both cases the view simply performs the usual hit tests on data to populate a bit array to share via a variable.

Specifying views without connecting them directly offers both **extensibility** and **modularity** in view development. D3 practitioners can take advantage of the view abstraction to create new views one at a time and coordinate them incrementally as needed to realize their designs. Design variations can be explored by quickly changing the set of available variables and their bindings to views being considered, even with little or no modification to D3 code from design to design. This depends on how well the aspects of appearance and behavior desired for variation are exposed as properties in a given view type. On the other hand, creation of new view types can readily exploit redundancy in code. The views in the examples all use code that is essentially the same beyond the D3 fragments. Code to define event handling such as for panning can be reused in the same way as in the world of D3 examples. Variant groups of view types may also serve to provide specific combinations of interactions for particular design cases. Curation of working sets of view types is likely to be an essential duty of users of the view abstraction.

View designers specify how live property values affect a view's appearance and behaviour and how interactions in a view affect live property values. The choice of live properties to use in a view design is flexible, yet often requires only a small set of quite sensible value types. Live properties also tend to be common across views, even quite different ones, and the separation of concerns in view design is sharp. Consequently, the **composability** of views is high.

A view can either be modified directly after the interaction occurs or it can be modified indirectly through the triggering of live property change. Direct modification will only occur on the view that the interaction is carrying out when there are indirect modifications through the live properties are carried out on all the views that are being coordinated. *Consistency* is maintained through indirect modifications, as these will visually look the same whether the event is carried out on a particular view or its coordinated views. Whereas direct modification of the view like the appearance of the rectangle on the brush event does not reflect on the other coordinated views.

By using the live properties architecture, we were able to decompose the complex coordinations between many connected visualizations into a simple set of properties. We were able to show how we used used selection and indication properties to produce many of the coordination patterns discussed above in an efficient manner. By decomposing interactions like panning and zooming into simple numerical values, we can define complex coordination patterns that are are *consonant*. Research is still needed to make the architecture support more comprehensive set of parameter types with live properties to be able to define complex and varieties of ways of coordination. We were also developing comprehensive set of D3 views to work with live properties. To achieve better coordination capabilities for both D3 and live properties. If they were able to decompose complex interactions into simple variables, then the architecture is readily able to integrate them into relevant properties and use them. Some behaviour's like selection and Hovering in D3 are straightforward as they involve simple "click" and "Hover" events of the mouse where as others like Drag and Zoom are not straightforward. Behaviours like Zoom in D3 are abstractions provided for panning and Zooming in D3. Decomposing this behaviour into simple set of translate and scale behaviours will require understanding of how these events function internally which is challenging. We were able to extract translate behaviour from D3 and use it for changing the Range property, whereas drag behaviour is something which we have not accomplished yet.

5 Conclusion

We present a work-in-progress on developing a view-level abstraction for coordinating multiple views in a single D3 visualization. Progress has been surprisingly smooth. Re-implementation of Live Properties for the browser was straightforward. Despite a moderate learning curve while implementing the base view wrapper and the various examples of common view types, actual difficulties interfacing with the D3 library have been few. Creating new view types to population of a core collection has accelerated rapidly after the first few; the most recent two (bubble chart and parallel coordinate plot) have taken only a few hours, i.e. "full D3 speed". The foremost shortcoming of the view abstraction, carried over from D3, is a lack of support for mixing navigation and selection interactions in any given view. Moving forward, we will explore migration of the view abstraction to serve as a CMV abstraction layer in Vega-Lite [17]. More broadly, we aim to support access to the full visualization data processing pipeline, starting with dynamic queries [2] and progressing though re-implementation of the full Coordinated Queries architecture in Improvise [22]. One possibility is to refactor Improvise into a client-server architecture for desktop, mobile, and web clients using the coordination architecture as the basis for remote including collaborative interaction via coordination [23].

Acknowledgments

We thank Remco Chang and Wenbo Tao for their insights. This work was supported by National Science Foundation Award #1351055.

REFERENCES

[1] C. Ahlberg. Spotfire: An information exploration environment. *SIGMOD Record*, 25(4):25–29, December 1996.

[2] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 619–626, Monterey, CA, May 1992. ACM.

[3] C. Ahlberg and E. Wistrand. IVEE: An information visualization & exploration environment. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 66–73, 142–143, Atlanta, GA, October 1995. IEEE.

[4] A. Aiken, J. Chen, M. Lin, M. Stonebraker, and A. Woodruff. The Tioga-2 database visualization environment. In *Proceedings of the IEEE Visualization '95 Workshop on Database Issues for Data Visualization*, pages 181–207, 1995.

[5] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[7] N. Boukhelifa and P. Rodgers. A model and software system for coordinated and multiple views in exploratory visualization. *Information Visualization*, 2(4):258–269, December 2003.

[8] A. Buja, D. Cook, and D. F. Swayne. Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99, 1996.

[9] H. Chen. Compound brushing. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 181–188, Seattle, WA, October 2003. IEEE Computer Society.

[10] J.-D. Fekete, P.-L. Hémery, T. Baudel, and J. Wood. Obvious: A meta-toolkit to encapsulate information visualization toolkits — one toolkit to bind them all. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 89–98, Providence, RI, October 2011. IEEE.

[11] R. D. Hill, T. Brinck, S. L. Rohall, J. F. Patterson, and W. Wilner. The Rendezvous architecture and language for constructing multiuser applications. *ACM Transactions on Visualization and Computer Graphics*, 1(2):81–125, June 1994.

[12] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: Integrated querying and visualization of large datasets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 301–312, Tucson, AZ, 1997. ACM.

[13] B. A. Myers, D. Giuse, R. B. Dannenberg, B. V. Zanden, D. Kosbie, E. Pervin, A. Mickish, and P. Marchal. Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer*, 23(11):71–85, November 1990.

[14] C. North and B. Shneiderman. A taxonomy of multiple window coordinations. Technical Report CS-TR-3854, University of Maryland Department of Computer Science, 1997.

[15] C. North and B. Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*, pages 128–135, New York, NY, USA, May 2000. ACM Press.

[16] S. F. Roth, P. Lucas, J. A. Senn, C. C. Gomberg, M. B. Burks, P. J. Stroffolino, J. A. Kolojejchick, and C. Dunmire. Visage: A user interface environment for exploring information. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 3–12, 116, San Francisco, CA, October 1996. IEEE.

[17] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, January 2017.

[18] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, January 2016.

[19] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, November 1994.

[20] Square. Crossfilter: Fast multidimensional filtering for coordinated views. http://square.github.io/crossfilter/, 2012.

[21] M. Takatsuka and M. Gahegan. GeoVISTA Studio: A codeless visual programming environment for geoscientific data analysis and visualization. *Computational Geoscience*, 28(10):1131–1144, December 2002.

[22] C. Weaver. Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 159–166, Austin, TX, October 2004. IEEE Computer Society.

[23] C. Weaver. Is coordination a means to collaboration? In *Proceedings of the International Conference on Coordinated & Multiple Views in Exploratory Visualization (CMV)*, pages 80–84, Zürich, Switzerland, July 2007. IEEE Computer Society.

[24] C. Weaver. Patterns of coordination in Improvise visualizations. In *Proceedings of SPIE (Visualization and Data Analysis)*, pages 1–12, San Jose, CA, January 2007. SPIE.

[25] C. Weaver. Cross-filtered views for multidimensional visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):192–204, March-April 2010.

[26] Worldometers.info. Covid-19 coronavirus pandemic. https://www.worldometers.info/coronavirus/, July 2020.

[27] J. S. Yi, Y. ah Kang, J. T. Stasko, and J. A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, November/December 2007.