University of California
Santa Barbara

Visualization Authoring for Data-driven Storytelling

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Donghao Ren

Committee in charge:

Professor Tobias Höllerer, Chair
Professor Matthew Turk
Professor George Legrady
Doctor Bongshin Lee

June 2019

The Dissertation of Donghao Ren is approved.

Professor Matthew Turk

Professor George Legrady

Doctor Bongshin Lee

Professor Tobias Höllerer, Committee Chair

March 2019

Visualization Authoring for Data-driven Storytelling

by

Donghao Ren

DEDICATE

TO

My Beloved Grandmother and Grandfather,

Guifen Liu and Tiesheng Ren.

Acknowledgements

First, I would like to express my special appreciation and thanks to my advisor Prof. Tobias Höllerer for mentoring me throughout the Ph.D. program. Tobias was always encouraging. His advice on both research and my career have been priceless. I would also like to thank my committee member Dr. Bongshin Lee, who offered me multiple internships at Microsoft Research and gave me invaluable guidance on visualization and general human-computer interaction. I also want to thank my committee member Prof. Matthew Turk who gave me helpful comments and feedback, and Prof. George Legrady who helped my development on aesthetics and visual language.

I am very much thankful to my collaborators Matthew Brehmer, Eun Kyoung Choe, Nathalie Henry Riche, Tibor Goldschwendt, and Yun-Suk Chang. This dissertation would not have been possible without their wonderful work.

I would also like to thank my undergraduate advisor Prof. Xiaoru Yuan who introduced me to the area of information visualization and guided me through several projects; and my mentor Saleema Amershi who introduced me to the field of interactive machine learning.

During my years at UCSB, I had the privilege to work with the AlloSphere research facility, a unique research instrument. I would like to thank Prof. JoAnn Kuchera-Morin, Dr. Andrés Cabrera, and Dr. Matthew Wright for their help and support with the AlloSphere.

I would like to thank my labmates. I received continued help and support from Ehsan Sayyad, Byungkyu (Jay) Kang, Peter Zhe Fu, James Schaffer, Brandon Huynh, Sikun Lin, Yi Ding, Adam Ibrahim, Christopher Hall, Benjamin Nuernberger, Kuo-Chin Lien, and Domagoj Baričević. I would also like to thank my friends at the Media Arts and Technology program, including Jieliang (Rodger) Luo, Jing Yan, Lu Liu, Han-Wei Kung, You-Jin Kim, Weidi Zhang, and Xiaoci Wu.

I am also thankful to my housemates and friends, especially Suoqing Ji, Bolun Wang, Rui Zhu, Shunyao Li, Bohan Li, Mengqian Liu, Bozhou Men, and Qingyun Liu.

Finally, thanks are also due to the following funding agencies: U.S. Army Research Office under MURI grant W911NF-09-1-0553; the Office of Naval Research (ONR) under contract N00014-14-1-0133, N00014-16-1-3002, and DURIP Instrumentation Grant N00014-13-1-0872; and National Science Foundation (NSF) under grant IIS-1748392.

Thank you to my girlfriend Cong Yan, for all her love and support.

Curriculum Vitæ
Donghao Ren

Education

2013 – 2019	Ph.D. in Computer Science, University of California, Santa Barbara
2009 – 2013	B.S. in Intelligence Science and Technology, Peking University

Research Experience

2013 – 2019	Research Assistant, University of California, Santa Barbara
Summer 2017	Research Intern, Microsoft Research, Redmond
Summer 2016	Research Intern, Microsoft Research, Redmond
Summer 2015	Research Intern, Microsoft Research, Redmond

Teaching Experience

Fall 2013	UCSB CS8 – Introduction to Computer Science
Fall 2017	UCSB CS184 – Introduction to Mobile Application Development

Publications

InfoVis'18	**Donghao Ren**, Bongshin Lee, and Matthew Brehmer, *Charticulator: Interactive Construction of Bespoke Chart Layouts*, *IEEE Transactions on Visualization and Computer Graphics (InfoVis'18), 25 (1), 2019* 🏆 **Best Paper Honorable Mention Award**
BELIV'18	**Donghao Ren**, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche, *Reflecting on the Evaluation of Visualization Authoring Systems*, *in Proceedings of BELIV 2018: Evaluation and Beyond — Methodological Approaches for Visualization*
VRST'18	**Donghao Ren**, Bongshin Lee, and Tobias Höllerer, *XRCreator: Interactive Construction of Immersive Data-driven Stories*, *in Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology (VRST'18) (Poster)*
Presence'18	Renate Häuslschmid, **Donghao Ren**, Florian Alt, Andreas Butz, and Tobias Höllerer, *Personalizing Content Presentation on Large 3D Head-Up Displays*, *PRESENCE: Virtual and Augmented Reality, 27 (1), 80–106, 2019*
PacificVis'17	**Donghao Ren**, Matthew Brehmer, Bongshin Lee, Tobias Höllerer, and Eun Kyoung Choe, *ChartAccent: Annotation for Data-Driven Storytelling*, *in Proceedings of IEEE Pacific Visualization (PacificVis'17), 230–239, 2017*

EuroVis'17 **Donghao Ren**, Bongshin Lee, and Tobias Höllerer, *Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering*, *Computer Graphics Forum (EuroVis'17), 36 (3), 179–188, 2017*

VINCI'17 Jieliang Luo, **Donghao Ren**, and George Legrady, *Anamorphic fluid: Exploring spatial organization and movements of images in a simulated fluid environment*, *in Proceedings of the 10th International Symposium on Visual Information Communication and Interaction, 2017*

VAST'16 **Donghao Ren**, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D. Williams, *Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers*, *IEEE Transactions on Visualization and Computer Graphics (VAST'16), 23 (1), 61–70, 2017*

VR'16 **Donghao Ren**, Tibor Goldschwendt, Yun Suk Chang, and Tobias Höllerer, *Evaluating Wide-Field-of-View Augmented Reality with Mixed Reality Simulation*, *in Proceedings of IEEE Virtual Reality (VR'16), 2016*

InfoVis'14 **Donghao Ren**, Tobias Höllerer, and Xiaoru Yuan, *iVisDesigner: Expressive Interactive Design of Information Visualizations*, *IEEE Transactions on Visualization and Computer Graphics (InfoVis'14), 20 (12), 2092–2101, 2014*

PacificVis'14 **Donghao Ren**, Xin Zhang, Zhenhuang Wang, Jing Li, and Xiaoru Yuan, *WeiboEvents: A Crowd Sourcing Weibo Visual Analytic System*, *in Proceedings of IEEE Pacific Visualization Symposium (PacificVis'14) Notes, Yokohama, Japan, 2014*

VAST'14 Xiaoru Yuan, Zhenhuang Wang, Zipeng Liu, Cong Guo, Hongwei Ai, and **Donghao Ren**, *Visualization of Social Media Flows with Interactively Identified Key Players*, *in Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST'14) (Poster), 291–292, 2014*

InfoVis'13 Xiaoru Yuan, **Donghao Ren**, Zuchao Wang, and Cong Guo, *Dimension Projection Matrix / Tree: Interactive Subspace Visual Exploration and Analysis of High Dimensional Data*, *IEEE Transactions on Visualization and Computer Graphics (InfoVis'13), 19 (12), 2625–2633, 2013*

Abstract

Visualization Authoring for Data-driven Storytelling

by

Donghao Ren

Data-driven storytelling is the process of communicating insights and findings that are supported by data, forming a visualization-based narrative. However, most current visualization creation tools either only support fixed sets of designs or require an in-depth understanding of programming concepts. To enable non-programmers to create custom visualizations for data-driven storytelling, we design interactions and implement user interfaces for visualization authoring. In the first part of this dissertation, we introduce and evaluate a series of three visualization authoring tools using traditional user interfaces: (1) iVisDesigner, which uses a data-flow model and enables users to author visualizations by specifying mappings from data to graphics interactively; (2) ChartAccent, a tool for annotating a given visualization; and (3) Charticulator, which allows users to design custom layouts interactively. We then reflect on the evaluation of visualization authoring user interfaces. In the second part of the dissertation, we extend our approach to multiple presentation media or display environments, including traditional 2-dimensional screens, large projection-based virtual-reality (VR) systems, and head-mounted virtual/augmented reality displays (HMDs). To leverage such immersive visualization environments, we ported and extended the iVisDesigner authoring approach to projection-based virtual reality. To facilitate the development of immersive visualizations, we built a visualization library called Stardust, which provides a familiar API to utilize GPU processing power in a cross-platform way. Finally, we present Idyll-MR, a system for authoring data-driven stories in virtual and augmented reality.

We evaluated these authoring tools and libraries, and demonstrated high expressiveness, usability, and performance, as well as portability across platforms. In summary, our contributions enable larger audiences to create visual data-driven stories using different presentation media, leading to an overall enriched diversity of visualization designs.

Contents

Chapter 1

Introduction

1.1 Motivation

Data-driven storytelling is the process of communicating insights and findings that are supported by data. Visualizations play an important role in data-driven storytelling. Highly customized visual representations of data that are tailored to the specific purposes and insights increases the effectiveness of communication. Currently, most visualization creation tools either only support fixed sets of designs or require an in-depth understanding of programming concepts. To enable non-programmers to create custom visualizations for data-driven storytelling, we design interactions, implement user interfaces, and develop programming libraries for visualization authoring.

Before discussing the visualization authoring tools, we first introduce the basic concepts of information visualization, visual analytics, data-driven storytelling, and visualization authoring.

Information Visualization Information visualization is the study of visual representations of data. S. Card *et al.* [1] define information visualization as "the use of computer-

supported, interactive, visual representations of abstract data to amplify cognition." Well-designed visualization shows data in meaningful ways and allows readers to gain insights. In this dissertation, unless specified otherwise, we also use word "chart" to refer to a data visualization.

There are two primary purposes of information visualization: analytics and presentation, as discussed below.

Visualization for Analytics Visual analytics a field that focus on facilitating analytical reasoning through the use of interactive visual interfaces [2]. Typically, a visual analytic user interface makes heavy use of visualization techniques, interaction paradigms such as linked views and dynamic queries, as well as data mining algorithms.

Visualization for Presentation Visualizations can also be used for presentation purposes. They are increasingly used among practitioners to tell stories that are supported by data. Researchers have explored the scope of visual data stories [3] or narrative visualizations [4]. In this dissertation, we follow Lee *et al.* [3]'s scope of *visual data stories*: a visual data story consists of a set of story pieces (findings or insights) that are supported by data; most of these pieces are visualized with appropriate annotation (*e.g.*, labels, pointers) or narration (*e.g.*, text) to clearly convey the message; and finally, these pieces are connected together in a meaningful way (spatial or temporal order) to support the overall presentation goal. The practice of *data-driven storytelling* is the act of using visual data stories for presentation. This dissertation focuses on designing visualizations for presentation purposes.

Visualization Authoring While visualizations are very useful for analytics and presentation, it is not easy to design and implement them. We use the term "visualization authoring" to denote the act of constructing information visualizations from data. Visualization

authoring can be done by drawing, programming, or using visualization construction tools [5]. Drawing visualizations, either on paper or using a computer-based drawing tool, is a tedious task because visualizations often contain many graphical elements (consider a scatterplot of a thousand points). On the other hand, programming a visualization requires the understanding of computer science concepts such as variables, functions, and loops. While visualization libraries (*e.g.*, ProtoVis [6] and D3 [7]) can drastically reduce the complexity of a visualization program, understanding the concepts in these libraries such as data mapping and selection still requires considerable programming experience. Visualization construction tools do not require programming. However, most existing tools (*e.g.*, Microsoft Excel) only support a fixed set of templates which prevent the design of novel visualizations. More powerful tools, such as Lyra [8], are often hard to learn. The primary focus of this dissertation is the design, implementation, and evaluation of visualization authoring tools.

Visualization in Virtual and Augmented Reality Virtual reality (VR) uses various display systems such as head-mounted displays (HMDs), projection-based environments (*e.g.*, CAVEs) combined with computer graphics to generate virtual environments. Augmented reality (AR), instead of creating a purely virtual environment, augment the real world with virtual content. VR/AR displays constitute a promising medium for information visualization because of the vastly available workspace (much more extensive than regular screens or even grids of regular screens), and the ability to display 3D objects and draw connections between objects (virtual and virtual, virtual and real, or real and real) in the workspace. Most current visualization authoring tools work on desktop environments and design 2D visualizations. In the second part of this dissertation, we explore visualization authoring for virtual and augmented reality environments.

Thesis Statement

> *A wide variety of information visualizations can be constructed with well-designed interactive interfaces without resorting to programming, thus providing a creative toolbox for data-driven storytelling. Authoring tools can also ease the cross-platform creation and deployment of visualizations, enabling data-driven storytelling in immersive environments.*

1.2 Outline

We attempt to prove the first part of this statement by designing, implementing, and evaluating multiple interactive visualization authoring tools. We address the second part by extending the tools we created to virtual and augmented reality and develop a system to facilitate the authoring of immersive data-driven visual stories.

This dissertation consists of three parts. In the first part, we address the problem of visualization authoring in desktop environments. We introduce a series of three visualization construction tools including iVisDesigner (Chapter 3), ChartAccent (Chapter 4), and Charticulator (Chapter 5). iVisDesigner allows users to construct visualizations by specifying data mappings and transformations in its user interface. ChartAccent focuses primarily on creating annotations on existing visualizations. Charticulator helps users specify custom layouts, and improves upon existing systems to support the creation of a large set of visualization designs without programming.

In the second part, we look into the authoring of visualizations in immersive environments including virtual and augmented reality (VR and AR). First, we propose a simple tool for creating immersive visualizations (Chapter 6), which was subsequently used in a user study that evaluates the effect of field of view (FOV) on the comprehension of wide-

field-of-view linked visualizations [9]. Then, we design a new visualization library called Stardust that allows users to write cross-medium visualization code in JavaScript (Chapter 7). Finally, we bring all the pieces together by building a new tool called Idyll-MR for immersive data-driven storytelling using visualizations and other graphical components (Chapter 8).

The third part concludes this dissertation by reflecting on the evaluation of visualization authoring systems, discussing the issues and limitations of current work, and presenting directions for future research.

1.3 Contributions and Impact

This dissertation makes the following contributions:

- New systems for interactively authoring (iVisDesigner and Charticulator) and annotating (ChartAccent) visualization, and evaluations of these systems by demonstrating the range of afforded designs and usability studies. In addition, we present a study of current annotation practice, and reflect on methods for evaluating for interactive visualization authoring systems.
- A scalable programming API and library (Stardust) for cross-platform visualization authoring, with its evaluation by efficiency measurements and demonstrations of supported platforms.
- A system for authoring immersive visual data-driven stories (Idyll-MR), and evaluations of the system through a demonstration of the range of afforded designs.

We have released iVisDesigner, ChartAccent, Stardust, and Charticulator to the public. The tools themselves will help the community of visualization designers, journalists, or

even the general public to communicate data better through custom bespoke visualization designs. We hope that our work will provide more visualization design possibilities for a broad audience.

1.4 Permissions and Attributions

1. The content of Chapter 3 is the result of a collaboration with Tobias Höllerer and Xiaoru Yuan. © 2014 IEEE. Reprinted, with permission from Donghao Ren, Tobias Höllerer, and Xiaoru Yuan, *iVisDesigner: Expressive interactive design of information visualizations*, *IEEE Transactions on Visualization and Computer Graphics* [10], Nov. 2014.

2. The content of Chapter 4 is the result of a collaboration with Matthew Brehmer, Bongshin Lee, Eun Kyoung Choe, and Tobias Höllerer. © 2017 IEEE. Reprinted, with permission from Donghao Ren, Matthew Brehmer, Bongshin Lee, Eun Kyoung Choe, and Tobias Höllerer, *ChartAccent: Annotation for data-driven storytelling*, *Proceedings of IEEE Pacific Visualization Symposium (PacificVis)* [11], Apr. 2017.

3. The content of Chapter 5 is the result of a collaboration with Bongshin Lee and Matthew Brehmer. © 2018 IEEE. Reprinted, with permission from Donghao Ren, Bongshin Lee, and Matthew Brehmer, *Charticulator: Interactive construction of bespoke chart layouts*, *IEEE Transactions on Visualization and Computer Graphics* [12], Aug. 2018.

4. The content of Chapter 7 is the result of a collaboration with Bongshin Lee and Tobias Höllerer, and has previously appeared in *Computer Graphics Forum* [13]. © 2017 Computer Graphics Forum, the Eurographics Association and John Wiley &

Sons Ltd. Published by John Wiley & Sons Ltd. Reprinted, with permission from the authors and the publisher.

5. The content of Chapter 9 is the result of a collaboration with Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche, and has previously appeared in *Proceedings of BELIV 2018: Evaluation and Beyond – Methodological Approaches for Visualization* [14]. Reprinted, with permissions from the authors.

Although I am the first author of all these previously published papers, the research presented in this dissertation is a result of close collaboration with my advisor Tobias Höllerer and co-authors including Bongshin Lee, Matthew Brehmer, Eun Kyoung Choe, and Nathalie Henry Riche.

Chapter 2

Related Work

This work builds on top of a rich research landscape of information visualization frameworks, toolkits, systems, as well as the understanding of virtual and augmented reality. We begin our review of related work by discussing influential theoretical background on information visualization and data-driven storytelling. We then give an overview on visualization programming frameworks and interactive visualization construction tools, annotation support, as well as visualization systems and user studies in immersive environments.

Information visualization is the study of visual representations of data. S. Card *et al.* [1] define information visualization as *"the use of computer-supported, interactive, visual representations of abstract data to amplify cognition."* Any expressive system facilitating flexible mappings of data to visual variables owes a lot to the semiological research of Bertin [15, 16]. Mackinlay [17] provided further foundation and presented automated tools for powerful modular visualization design. Shneiderman [18] analyzed the data types and tasks in information visualization, and presented a taxonomy for them. Card *et al.* [19] organized previous visualization designs, and presented categorization of data types and visual mappings. Information visualizations can be used for either presentation

or analytical purposes. In this dissertation, we primarily focus on visualizations for presentation purposes. This leads to the following discussion about data-driven storytelling.

2.1 Data-driven Storytelling

Visual data-driven storytelling often involves the use of information visualization techniques to support or complement a written or spoken narrative. Segel and Heer [4] refer to this form of storytelling as *narrative visualization*, and in their proposed design space of narrative visualization, they indicate the importance of textual and graphical annotation as well as visual highlighting. Kosara and Mackinlay [20] have emphasized the importance of annotation and highlighting in visual data-driven storytelling, particularly in the context of live presentations. In an effort to scope visual data stories, Lee *et al.* [3] emphasize the need of the intended message in stories and the role of written explanations or annotations that help the viewer capture the message. They also identify that, to make it easier to tell data-driven stories, it is useful to support the easy creation of custom annotations through direct manipulation and reuse of existing story elements. Stolper *et al.* [21] identify seven common annotation techniques used to communicate narrative and explain data in popular data-driven stories. In addition, Choe *et al.* [22] provide empirical evidence of the important role of annotation through an analysis of the design choices used in live presentations by self-tracking enthusiasts or "quantified selfers." Hullman and Diakopoulos [23] consider annotation to be one of the four editorial layers in this genre of storytelling, alongside the layers of data, visual representation, and interactivity; and while there are certainly a number of tools and techniques that address these other layers, support for the annotation layer remains underdeveloped. Boy *et al.* [24] studied the engagement factor in data-driven storytelling; Ma *et al.* [25] explored storytelling in scientific visualizations.

Our work is informed by prior research on visualization theories and models, as well as research on data-driven storytelling. In our demonstrations and evaluations of iVisDesigner and Charticulator, we highlight its coverage of the information visualization design space. To guide our design of ChartAccent, we conduct a survey of current annotation practices and summarized the findings as a design space of annotations. We use Lee *et al.* [3]'s scope of visual data stories to guide the design of Idyll-MR.

2.2 Visualization Authoring Systems and Toolkits

There is a variety of existing interactive tools and programming frameworks for visualization authoring. In this section we discuss the related programming frameworks and languages, interactive visualization authoring tools for visualization authoring. We also discuss frameworks and tools for data-driven storytelling purposes.

2.2.1 Programming Frameworks and Languages

There is a large collection of programming frameworks, and languages for simplifying visualization programming.

Programming toolkits General-purpose drawing APIs such as OpenGL, HTML5 Canvas, and Processing [26] define programming interfaces to draw low-level elements. Even for experienced programmers, it is a tedious process to create visualizations with these APIs. Thus, visualization frameworks have been created for better abstraction. The InfoVis Toolkit [27] and Improvise [28] provide a set of basic widgets. Chi *et al.* [29] proposed a spreadsheet approach. "behaviorism" [30] uses three graphs to represent dynamic visualizations. To create novel designs, users need to create new widgets or inherit existing ones. Heer *et al.* proposed Prefuse [31, 32], a toolkit for interactive visualization. It first

transforms abstract data into visualizable form by a *filtering* process, and renders the visualizable form by using a *view* process. It allows for advanced integration of existing operators to create novel techniques, but typically users will need to define new operators in the process.

Declarative models and languages Declarative models and languages for information visualization have been presented, including the Grammar of Graphics and its implementation ggplot in R [33, 34]. ProtoVis [35, 6] provides a declarative language for information visualization, designed and implemented in Java and JavaScript. Bostock *et al.* designed D3.js [7], a JavaScript library for creating web-based visualization designs. It facilitates the manipulation of DOM elements with data. Vega [36] employs a reactive data-flow model and support visualization by adding a variety of support modules, such as scales and layout transformations. Vega-Lite [37] is a high-level declarative language that compiles to Vega. All of these programming frameworks require users to write computer programs or formal specifications to create visualization designs.

GPU-based visualization rendering GPU-based rendering has been widely used in scientific visualization tools. For instance, GPUs are used for accelerating volume rendering [38], particle and glyph rendering [39]. The Visualization Toolkit [40] uses GPUs extensively for scientific visualizations. However, most of these techniques are limited to data that has existing 2D/3D spatial structures. Customization of visual encoding and data binding is limited to the domain of usage. Our work focuses on information visualization where there can be a wide variety of visual mapping and encoding, and existing spatial representations are not guaranteed to exist.

For information visualization, several research projects recently utilized GPUs for accelerating visualizations. For example, SplatterJs [41] provides extensive support for

scatterplots with WebGL rendering techniques and the FluidDiagrams [42] system implements bar charts, line charts, parallel coordinates, and cone tree visualizations [43]. SandDance [44] uses WebGL to efficiently render instance-based visualizations for large datasets and enable smooth transitions between layouts. imMens [45] uses the GPU for efficient data transformation and rendering, but it does not allow customization of GPU-rendered graphical marks. GPU acceleration has also been successfully applied for edge bundling [46]. These visualizations are custom-made, and thus developers of such systems have to write GPU shaders, convert data to buffers, and execute rendering commands. They also provide fixed sets of visualization templates or modules instead of providing developers flexibility in creating their own visualizations.

While GPU-based techniques can significantly improve performance, they are challenging to implement and thus not accessible to developers without expertise in the computer graphics pipeline and shader programming. While compilers (*e.g.*, Sh [47] and Brook [48]) can make GPU acceleration accessible through common programming languages such as C/C++, they are not designed to support information visualizations. In Chapter 7, we present Stardust. It provides visualization developers a simple and familiar programming interface to leverage GPU processing power. We anticipate that developers with D3 expertise could easily adopt Stardust.

2.2.2 Interactive Visualization Authoring Tools

Both the research community and the industry have explored the space of interactive visualization authoring tools. These tools aims at enabling non-programmers to construct information visualizations.

General-purpose graphics tools General-purpose vector graphics software, such as Inkscape and Adobe Illustrator, is widely used for graphical design, and we drew some

inspiration for our user interface from such products. While it is possible to create visualizations with general-purpose design tools, they are not specifically designed for creating visualizations. For example, there is no support for parameterizable mapping from data to graphics and non-trivial layouts. Graphical items have to be constructed and positioned individually.

In the following paragraphs we discuss different approaches for interactive visualization construction user interfaces. Our discussion follows the categorization from Grammel et al.'s survey [49] of visualization construction user interfaces.

Dataflow systems Dataflow-based systems, such as ConMan [50], AVS [51], IRIS Explorer [52], and VisTrails [53] use a pipelined approach and flow diagrams to represent the progress from data to visualization. These systems are particularly good at defining data transformations, but not very flexible for defining interlinked mappings from data to graphical elements. In addition, pipeline-based systems focus on representing the pipeline itself, there is little screen estate left for displaying and editing the visualizations. Our work provides an integrated representation and manipulation of graphical mappings, with all graphical elements created, presented and edited in the same canvas which dominates the user interface, allowing for better understanding of the overall visualization design.

Template editors A lot of commercial software has the functionality to interactively create visualization designs for data, for example the chart feature in Microsoft Excel and similar spreadsheet software, various chart creation tools such as Plot.ly [54], ChartBlocks [55], RAWGraphs [56], and Quadrigram [57]. Many web-based systems, for instance ManyEyes [58], Sense.us [59], or CommentSpace [60], include visualization creation user interfaces to support parts of their workflow. Although some of these systems

provide richer sets of customizable options, they all follow the template editor approach. In this approach, the author selects data and chooses a predefined visualization template (*e.g.*, bar chart, line chart, etc.), and then configures the template by adjusting its options in a panel. This widely used approach clearly lacks expressivity — users are unable to create custom visualizations that are not covered by the set of tempaltes.

Shelf-configuration Shelf-configuration [49] is an alternative approach to template editors. In such user interfaces, an author maps data attributes to encoding channels of marks, such as x, y, color, size, and shape. The system synthesize the visualization based on its internal rules. For instance, to create a scatterplot, the author chooses the circle mark type, and maps two numerical data attributes to the x and y encoding channel. To color the scatterplot by a third categorical data attribute, the author maps the data attribute to the color encoding channel. Tableau Desktop [61] is a highly sophisticated state-of-the-art commercial visualization system following the shelf-configuration method. It provides good flexibility for visualization designs. While the shelf-configuration approach offers more expressivity, it still lacks fine-grained control and flexible combination or layout of graphical elements.

Visual builders Visual builders take a lower-level approach. In such user interfaces, authors can manipulate basic graphical elements, bind data to them, and configure their appearances. For multivariate data, Claessen *et al.* [62] allow users to position axes and put scatterplots and links between them interactively. However, it only supports multivariate data and axis-based visualization designs. Sketch-based interactions, like in SketchStory [63] and SketchInsight [64] have been explored. While sketch-based interactions are very intuitive, these systems currently only support a very limited set of designs. Bret Victor presented a tool [5] that allows users to define drawing procedures with geo-

metrical constraints, which are parametrized in an interactive canvas. It requires procedural thinking, where users define loops to draw simple visualizations such as a scatterplot. In Chapter 3, we present iVisDesigner, which uses a declarative approach, where loops are defined implicitly by *data selectors*. SageBrush [65] uses "partial prototypes" to define spatial properties for "graphemes", and supports editing of primitive properties. iVisDesigner expands on this theme by enabling data transformation and generation, and supporting interaction with designed visualizations. Lyra [8] is a very recent addition to the interactive visualization design landscape. Based on the JSON-based declarative visualization grammar Vega [66], it allows users to define visualizations interactively by constructing scales, guides and marks. Sophisticated layouts and transformations are enabled via transformation pipelines. Lyra and Vega only operate on tabular datasets, while our work iVisDesigner also supports hierarchical datasets with a fixed schema and references between data items. Lyra is more oriented towards designing a single piece of visualization, while our system focuses on canvases that allow users to draw and link different designs. Furthermore, iVisDesigner supports designing basic interactions such as brushing and linking. To ease the creation of marks from data items, Data Illustrator [67] uses a repetition and partition operator for multiplying marks. However, the repetition and partition grid can handle only simple layouts such as grid and horizontal and vertical stacking, and these layouts only work with primitive shapes. To create layouts with a complex glyph, chart creators have to manually overlay multiple groups of marks that are individually laid out. VisComposer [68] combines visual programming, textual programming, direct manipulation, and surrogate objects for chart creation. Different visual encoding design choices can be arranged in a scene graph editor to create complex charts. However, VisComposer supports only a fixed set of compositions and requires programming to create additional visual encodings. In Chapter 5, we present Charticulator, which prioritizes layout as a deliberate design choice. With multiple levels

of partial layout specifications, chart creators can construct a wide range of chart layouts. Charticulator also supports layouts incorporating several alternative coordinate systems. Furthermore, Charticulator allows chart creators to export their designs into reusable chart templates which can then be imported into other visualization systems such as Microsoft Power BI (Figure 5.9).

2.2.3 Frameworks and Tools for Data-driven Storytelling

Beyond authoring a single visualization, there are multiple tools to support the data-driven storytelling practice. For example, Ellipsis [69] supports the easy creation of data-driven stories by combining programming and the configuration of annotations, parameters, and scenes. DataClips [70] supports the authoring of data-driven videos. Timeline Storyteller [71] enables people to author and present expressive data-driven stories with time-based data (*e.g.*, sequences of events).

Annotation support Here we have a specific look into annotations because they are critical to data-driven storytelling. Many contemporary visualization tools provide limited annotation support, particularly in the domain of business intelligence [72]. Annotation functionality also features prominently in tools intended for collaborative visual analysis, such as ManyEyes [58], sense.us [59], and CommentSpace [60], as the ability to annotate a chart allows people to share their insights with others. In the following, we give further consideration to tools that support data-driven or data-aware annotation, and to annotation support in interactive visualization authoring tools.

In the context of visualization, annotation can mean more than the mere addition of textual and graphical elements to an existing chart. Annotation functionality can be implemented in such a way that annotations are aware of the data being visualized. Heer and Shneiderman's refer to "data-aware annotations" [73], or graphical and textual

elements that appear in response to the interactive selection and brushing of visual marks that correspond to data [74]. This concept is realized in Click2Annotate [75] and Touch2Annotate [76], in which the semi-automated annotation of data items is facilitated via simple direct manipulation interactions. In a prototype tool by Heer *et al.* [74], a set of annotations for data items can appear in response to query criteria as well as selection and brushing interactions, and their tool decides which annotations to show based upon a relevance ranking.

Kandogan [77] introduced the concept of *just-in-time annotation*, in which a chart is automatically annotated as a person interacts with it. Similarly, Bryan *et al.* [78] introduced an approach for automatically annotating charts depicting time-oriented data during interactive exploration based on visual salience and significant features in the data. Altogether, these data-driven approaches certainly eliminate the tedium of manually annotating data items in a chart, though it could also be argued that these approaches take too much control away from the person using the system: they neither provide full control over which data items are annotated, nor do they provide control over the form or content of these annotations. Moreover, these approaches are predominantly geared toward data analysis tasks, and while many annotations in the context of visualization do correspond with specific data-bound graphical elements, every annotation in a chart should not be required to directly reference the underlying data. In Chapter 4, we present ChartAccent, which focuses more on the task of augmenting a chart for communication, combining both data-driven and manual annotation. With ChartAccent, we provide more freedom in terms of deciding which items get annotated and in terms of the form and content of these annotations, leveraging a data-driven approach when needed. Finally, while data-aware annotations that are triggered by interactive selection certainly fall under our purview, we do not wish to downplay the importance of static (*i.e.*, non-interactive) forms of annotation when presenting a chart to an audience. This view is reflected by

The New York Times' deputy graphics director Archie Tse, who recently argued that the best form of visual storytelling is often *static* [79].

Many recent interactive visualization authoring environments or visualization construction interfaces [49, 80] attempt to balance simplicity with an expressive range of visual encoding choices. However, they generally lack a support for annotation — especially data-driven annotation. With general-purpose chart authoring tools such as those offered in Microsoft Excel, chart authors are limited to selecting individual items or data series and modifying their visual encoding; data-driven operations such as selecting and annotating data items within a particular value range, for example, are not supported. As a result, an author will typically create a chart with a tool such as Excel and subsequently export it to a graphical editing tool such as Adobe Creative Suite to manually add annotations [81, 82], and in doing so awareness of the underlying data is lost.

Recent interactive visualization authoring tools such as Lyra [8] and our work iVisDesigner are more expressive than general-purpose tools like Excel in terms of visual encoding design choices. Although authors can create data-driven annotations with these tools, their learning curve is significant, and the annotation process is particularly tedious. We also draw inspiration from Mr. Chartmaker [83], the interactive charting tool used internally at *The New York Times*, which provides some data-driven annotation support via direct manipulation and selection of individual data items; we build upon these capabilities in ChartAccent by introducing data-driven selections in addition to annotation via manual selection.

Tableau Desktop [61] also provides several options for annotating charts, some of which overlap with ChartAccent. For example, Tableau allows a person to add trend lines to a chart, and text annotations of marks in Tableau can be data-driven, referencing marks' underlying data attributes. However, ChartAccent goes beyond Tableau in various aspects. In Tableau, three annotation targets are explicitly denoted when interacting with

a chart: a *x*, *y point*, a rectangular *area*, or any selected *mark(s)*. With ChartAccent, we consider a wider set of annotation targets informed by our survey of 106 annotated charts, including both one- and two-dimensional ranges and other targets listed in Section 4.2.2. In addition, highlighting marks in ChartAccent is simpler and more flexible than Tableau, without requiring explicit set creation; selected marks can be highlighted directly without affecting unselected marks.

It is certainly possible to author a chart with precise control over annotation using visualization programming libraries and packages such as ggplot2 [34] or D3.js [7], or using libraries specific to data-driven annotation and labelling such as swoopyDrag.js [84] or labella.js [85], or by using the Hanpuku D3-Illustrator bridge tool [82]. With ggplot2 in particular, chart authors can add sophisticated data-driven annotations such as trend lines or annotations relying upon more complex statistical models. However, although these approaches offer precise control, they require programming skills and can only provide asynchronous feedback to the author; with ChartAccent, we opt for an interactive visualization authoring environment that requires no specialized programming knowledge and provides instant feedback to the author.

2.3 Constraint-based Authoring

Prior work has incorporated mathematical constraints for designing user interfaces, diagrams, and some forms of graphs. An early example is ThingLab [86], which involves connecting geometrical objects (*e.g.*, lines, points) using constraints via mouse and keyboard interactions. Later, Fogarty and Hudson created the GADGET toolkit [87] to incorporate numerical optimization for user interface systems. Beyond research prototypes, the Android development ecosystem recently introduced ConstraintLayout [88] as an option to specify widget positioning in user interfaces: an interactive constraint editor

in Android Studio facilitates constraint authoring by allowing developers to specify constraints via drag-and-drop interactions. However, these existing interactive constraint-based authoring approaches only support a fixed set of objects or widgets.

Constraint-based approaches have also been applied to data visualization. GLIDE [89] is a constraint-based graph visualization system that allows people to specify *visual organization features* such as symmetry, alignment, or clustering, whereupon these features are translated to geometrical constraints. Delaunay [90] supports similar constraints for visualizing hierarchical data. The TRIP system [91] employs a constraint-based approach for visualization and animation, where the constraints are specified as a *visual structure representation* in Prolog. These existing approaches either focus on graph or tree visualizations, or they use programming languages for constraint specification. With Charticulator, we set out to enable the creation of bespoke charts without programming, so we investigated ways to efficiently specify common layout constraints via a set of well-designed user interactions and by incorporating data binding into these constraints.

2.4 InfoVis. in Immersive Environments

The second part of this dissertation is about visualization authoring for immersive environments. In this section, we discuss the related work on this topic.

2.4.1 Understanding Immersive Factors

It is important to consider the results of previous experiments on immersion factors [92], as they can provide valuable hints for designing information presentations in VR/AR experiments. In this thesis we mainly consider the factor of field of view (FOV), which is critical for visualizations that attempt to utilite a vast virtual space and connect data together through visual links. Arthur [93] studied the effects of FOV on task perfor-

mance and participant comfort. Covelli *et al.* [94] experimented FOV in pilot performance in flight training scenarios. Jones *et al.* [95] found that in medium-field VR, peripheral vision is an important source of information for the calibration of movements. Ragan *et al.* [96] examined the effects of varying FOV in terms of a visual scanning task as part of virtual reality training. Higher field of view led to better training performance. The results generally indicate that higher FOV leads to better performance. However, Knapp *et al.* [97] found that limited FOV of HMDs is not the cause of distance underestimation in VR. Ball *et al.* [98] showed that the opportunity for physical navigation is more important than increased FOV for display walls.

Tracking artifacts including latency and jitter are also of research interest in virtual reality. Ventura *et al.* [99] studied the effects of tracker reliability and field of view on a target following task using an HMD as the simulator. This study examined the same overall types of immersion parameters as our work here, but wide-FOV AR was not considered, as the maximum FOV was limited by the simulator HMD, and tracking artifacts were limited to interruptions of varying length. A low FOV proved detrimental to user performance, as did prolonged sensor dropout periods. Buker *et al.* [100] studied the effect of latency on a see-through HMD and determined that simulator sickness could be reduced by predictive tracking compensation.

Our understanding of FOV in AR is still limited [101]. Researchers have explored the effect of field of view with actual AR prototype devices. Van Nguyen *et al.* [102] developed a wide FOV AR display. Using a similar setup, Kishishita *et al.* found that a distribution of annotations in the peripheral vision decreases target discovery rate [101], and task-performance differences between in-view and in-situ annotations become smaller as the FOV approaches 100 degrees and beyond [103]. Although they explored wide FOVs, their annotations are relatively small and point-based, compared to what we used in our experiment.

Conducting AR user studies is challenging. With AR technology still far from a standardized technology, display devices suffer from low resolution, low field of view, latency, improper occlusion, and other perceptual problems [104]; the capabilities of devices differ a lot from one to another, and many AR immersion parameters are currently not achievable, especially in the mobile AR domain [105].

Usability evaluations in AR have been performed regarding questions of perception, performance and collaboration [106] using a range of evaluation techniques [107, 108]. In our experiment, we employ objective measurements on task time, correctness and head movement speed, subjective measurements from pre- and post-study questionnaires and informal feedback from participants.

There are currently no commercial solutions yet for truly wide FOV mobile AR, even though it has been the focus of both academic and industrial research [109, 110, 111]. Likewise, low latency and high-precision tracking is still a challenging problem, particularly for mobile AR. Impressive progress has been made at some cost of hardware bulk and expense [112].

Immersive Visualization Researchers have also recently developed tools specifically to help people create more expressive chart designs (*e.g.*, Data-Driven Guides [113], Lyra [8], Data Illustrator [67], InfoNice [114], Charticulator [12]) or easily augment charts with manual and data-driven annotations (ChartAccent [11]). However, these authoring systems are designed for traditional desktop environments. While several immersive data-driven visualization systems have been demonstrated by now [115, 116, 117] and Isenberg *et al.* recently discussed opportunities and challenges for immersion in visual data-driven stories, there has been limited research on how to facilitate the creation of immersive data-driven stories [118].

Researchers have recently paid more attention to the exploration of immersive data

exploration. Donalek *et al.* [119] designed a system for collaborative immersive visualization with basic visualizations for scientific data. ViSTA Widgets [120] provides reusable building blocks for 3D user interfaces. ImAxes [121] enables people to explore multivariate data by manipulating axes in a virtual environment and exploring the visualizations that emerge from various combinations of axes. Le *et al.* [122] proposed a system that supports distributed (involving remote users) creative collaboration. DXR [116] enables both interaction of and programmatic construction of 2D/3D visualizations in Unity using a declarative framework inspired by Vega-Lite [37]. IATK [117] allows a Unity user to create complex linked visualizations in immersive environments, however, it doesn't yet provide direct support for collaboration or constructs for data-driven storytelling. Dataspace [123] enables a hybrid immersive analytics scenario, but it does not support authoring extensively.

Researcher have also conducted empirical studies to compare different techniques designed for immersive data analytics. For example, Cordeil *et al.* [124] compared HMD- and CAVE-style virtual reality in network analysis. Bach *et al.* [125] compared three visualization environments — a desktop with mouse & keyboard against the tangible input with Microsoft HoloLens and with a handheld tablet — with 3D scatterplots for four tasks. Yang *et al.*, [126] reported three studies conducted to compare different spatial encodings of global flow networks for immersive analysis. Dataspace [123] enables a hybrid immersive analytics scenario, but it does not support authoring extensively.

Although many tools exist to support the authoring of data-driven stories or immersive analytics, to our knowledge, currently none of the existing authoring systems support the creation of immersive data-driven stories.

Similarly, there is a paucity of systems that allow for cross-platform authoring of data visualizations or data stories. Our system, Idyll-MR, enables authors to create data-driven visualizations and stories that can be experienced on different conventional and

immersive platforms: web, virtual reality, and augmented reality. Authoring can also occur on any of these platforms, or, for best usability, on a combination of platforms, and even by a collaborative team of authors with different media competences.

2.4.2 Rendering in Immersive Environments

Researchers have explored large display walls [127] and immersive environments such as CAVEs [128]. The growth of virtual and augmented reality suggests a stronger need for supporting a wide variety of display environments, and efficient rendering is of particular importance in these environments. Although more research is needed to understand when 3D and 2D visualizations are preferable in these environments, enabling more designers to support them can facilitate the exploration of the design space and thus improve our understanding about it.

There are many libraries, frameworks, and engines that support rendering in Virtual Reality (VR) and Augmented Reality (AR), such as FreeVR [129], OpenSG [130], Processing [26], TechViz [131], Unreal Engine [132], and Unity Game Engine [133]. However, there is little support for building complex information visualizations. Developers have to manually map data items to graphical marks. In a recent demo, Le *et al.* introduced b3.js [134], which supports bar charts, scatterplots, and surface plots in VR displays such as the Oculus Rift. However, it only supports a fixed set of visualizations, and depends on many other libraries. Stardust, in contrast, focuses on solving the data binding problem. It does not provide a set of predefined visualizations, but exposes an easy-to-use API for specifying and manipulating GPU-rendered marks.

Different display environments have different ways of content rendering. For example, some stereoscopic displays employ tilted view frustums; full-surround spherical displays use Omnistereo [135]. We designed the Stardust's core API to be platform-independent

thus allowing developers to customize the platform-dependent part. In our current version, we provide support for WebGL in normal 2D, 3D, and WebVR. By integrating with its rendering library, Stardust visualizations also can be rendered in the AlloSphere [136], a full-surround spherical multi-projector display environment. With these display environments we have supported, we demonstrate a web-based coding playground that allows developers to create visualizations using Stardust and run them without modification in multiple display environments including WebGL, Cardboard, HTC Vive, and the AlloSphere.

Part I

Interactive Visualization Authoring (Canvas-Based)

In this part of the dissertation, we present the design, implementation, and evaluation of three interactive visualization authoring tools including iVisDesigner, ChartAccent, and Charticulator. The three tools focus on three important aspects of visualization authoring for data-driven storytelling: data mapping, annotation, and layout, respectively.

iVisDesigner We first present iVisDesigner, a web-based user interface that enables users to interactively design information visualizations for heterogeneous datasets without the need for textual programming. iVisDesigner uses a data-flow framework to represent the visualization design. The user maps data items to graphical elements by choosing a data collection, and (optionally) specify filters in a property panel. Attribute mappings are specified in the property panel and a style panel. iVisDesigner also supports commonly used transformations, such as aggregation, formula-based data transformation, force-directed graph layout. These transformations are represented as nodes in the data-flow graph and exposed as extra data items and/or attributes which can be mapped to graphical attributes. Finally, iVisDesigner also enables a visualization designer to create end-user interactions such as range selection and item dragging. We demonstrate iVisDesigner's expressiveness through a set of example visualizations, and evaluate its usability with user evaluation.

ChartAccent iVisDesigner is a tool for visualization construction. However, simply presenting raw charts is not sufficient for data-driven storytelling because the presenter also needs to create annotations, such as arrows and highlights to direct attention, numbering to facilitate the narrative, and value lines, texts, and images to provide contextual information. In Chapter 4, we first conduct a survey about current annotation practice. Based on the resulting design space of annotations, we design ChartAccent, a tool to annotate existing charts. We demonstrate ChartAccent's effectiveness by presenting usage

scenarios and evaluate its usability through a controlled study where we let participants reproduce annotations with ChartAccent.

Charticulator iVisDesigner and similar tools such as Lyra [8] use a data-flow model which enforces a uni-directional mapping from data to graphics. This model well captures the traditional concept of visual mappings, such as mapping data to x, y, *width*, *height* values. However, data mapping alone cannot directly represent a wide range of charts. Layout transformations are often required. For example, in a basic horizontal bar chart, we need to figure out the widths and x positions of the bars. The usual way is to specify layout computations such as $x_i = left + i \times (width + step)$. More complex formulas will be required to support multi-column bar charts, stacked bar charts, or full-width stacked bar charts. Data-flow systems often resort to layout transformation modules, each module performs a specific layout. In Chapter 5, we present a new tool called Charticulator, which focus on designing expressive layouts. Charticulator transforms a chart specification into mathematical layout constraints and automatically computes a set of layout attributes using a constraint-solving algorithm to realize the chart. It allows for the articulation of compound marks or glyphs as well as links between these glyphs, all without requiring any coding or knowledge of constraint satisfaction. Furthermore, thanks to the constraint-based layout approach, Charticulator can export chart designs into reusable templates that can be imported into other visualization tools. In addition to describing Charticulator's conceptual framework and design, we present three forms of evaluation: a gallery to illustrate its expressiveness, a user study to verify its usability, and a click-count comparison between Charticulator and three existing tools. Finally, we discuss the limitations and potentials of Charticulator as well as directions for future research.

Charticulator is the culmination of the three tools. The framework of Charticulator

can support most features from iVisDesigner (except the ability to design end-user interactions), and allow for many visualizations that are not possible or very tedious to create with iVisDesigner. In addition, it is theoretically possible to use Charticulator output as input to ChartAccent.

Chapter 3

iVisDesigner: Interactive Design of Information Visualizations

3.1 Introduction

Programming frameworks for information visualization such as Prefuse [31], Proto-Vis [35] or D3.js [7] provide very useful abstractions for visualization designs that make programming easier and more elegant. Frameworks can utilize existing programming languages, such as JavaScript in the case of D3.js, or new programming languages, as in the case of Processing. However, they all require textual programming, which limits use to a population of coders, or otherwise imposes a fairly steep learning curve. Most of these frameworks require iterating back and forth between programming and execution stage, and thus adjustment of parameters can be cumbersome.

On the other hand, some information visualization toolkits support interactive ways

The contents of this chapter have been previously published in *IEEE Transactions on Visualization and Computer Graphics*. © 2014 IEEE. Reprinted, with permission from Donghao Ren, Tobias Höllerer, and Xiaoru Yuan, *iVisDesigner: Expressive interactive design of information visualizations*, *IEEE Transactions on Visualization and Computer Graphics* [10], Nov. 2014.

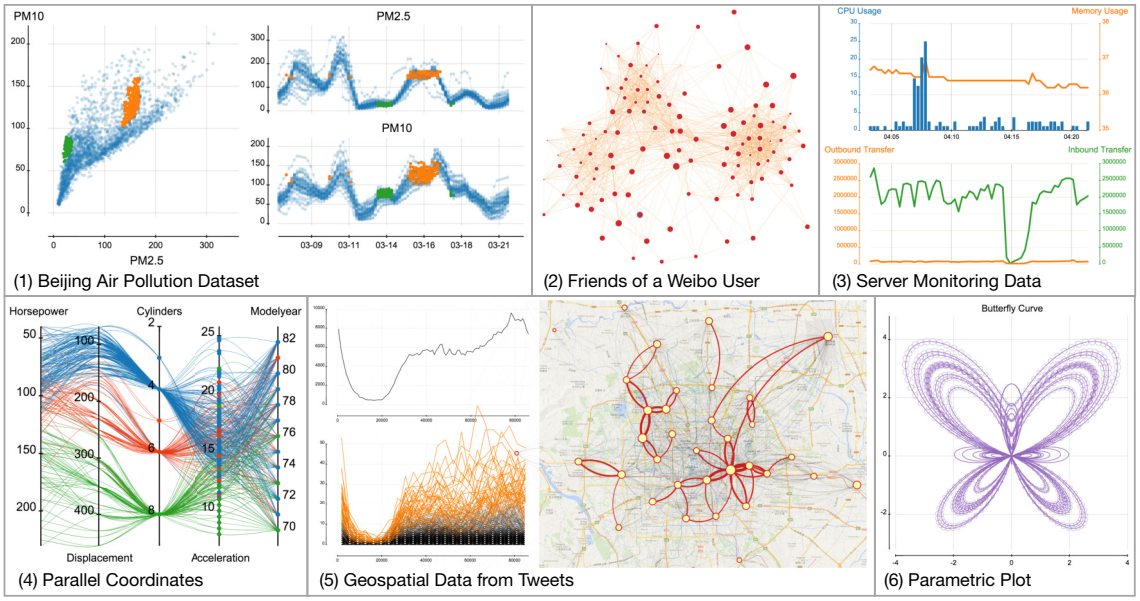


Figure 3.1: Example visualization designs created in iVisDesigner.

of creating visual designs [49], from early pipeline-based systems [51] to more unified approaches [62, 137, 8]. These systems are easy to use, and generally offer a What You See Is What You Get editing experience, which greatly assists parameter tuning. However, compared to textual programming, they are generally less expressive. For example, the Flexible Linked Axes toolkit [62], which inspired parts of our visualization functionality, only covers axis-based designs for multidimensional data visualization, arguably a small portion of the whole design space. There is a need for highly flexible toolkits that support a wide spectrum of visualization designs.

In this chapter, we present iVisDesigner, a web-based system that enables users to design information visualizations for heterogeneous datasets interactively, without the need for textual programming. Compared with other approaches such as Flexible Linked Axes [62] and Gold [138], and commercial software such as Microsoft Excel and Tableau, iVisDesigner focuses on expressiveness and modular visualization design flexibility, covering a wider range of the information visualization design space.

Expressiveness in iVisDesigner is supported by its underlying framework and user interaction provisions. The framework utilizes a flexible internal representation of visualization designs, which is carefully exposed in a unified user interface. Users specify designs via a mouse or pen-based user interface in a web browser, utilizing drag and drop, sketching, and context menu elements. The system supports visual analytics tasks, such as brushing and linking, and visualization customizations, both during visualization design and interactive exploration of data in completed designs. Users can embed the designed visualizations into existing websites or web-based applications by inserting a piece of JavaScript provided by iVisDesigner.

The main contribution of this work is an expressive framework to represent visualization designs for different types of data, allowing users to interactively arrange visual elements in different ways, combining and linking different types of visualizations. We discuss our design decisions, implementation choices, as well as system limitations, and demonstrate the expressiveness of our system by presenting a variety of example applications on different types of datasets. We provide evaluation of our system in form of a performance analysis and an informal user study. Our prototype system exhibits high expressiveness compared with existing systems, while maintaining good performance and usability.

The chapter is organized as follows: We first present the design of the framework and user interaction, followed by notable implementation details. Next we discuss example applications to exemplify coverage of the information visualization design space. We then present system evaluation in the form of performance measurements and an informal user study. Finally, we discuss overall results and limitations.

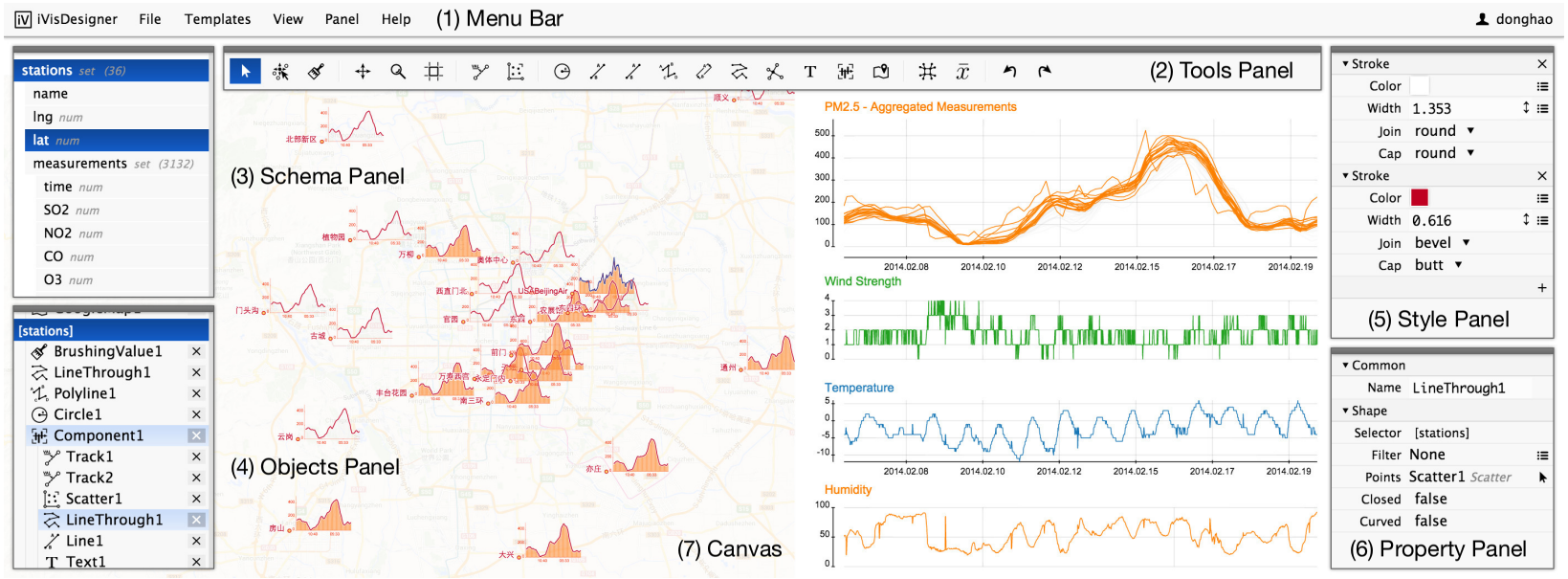


Figure 3.2: The interface of iVisDesigner. (1) Menu Bar: Commands for loading or saving visualization designs, view settings, login and logout. (2) Tools Panel: Tools for moving objects around, creating new objects and changing the view. Grouped into different categories. (3) Schema Panel: Shows the structure of the dataset. Allows selection. (4) Objects Panel: Shows the objects currently in the visualization design. Allows selection. (5) Style Panel: Adjust graphical styles for currently selected objects. (6) Property Panel: Adjust properties of currently selected objects. (7) The canvas to draw the visualization. In this example, a visualization of Beijing Air Pollution data is presented. There are two linked views, the left view shows the timeline plots of PM2.5 indexes for each measurement station on top of a map, the right plots shows the trends of the PM2.5 indexes, wind strength, temperature and humidity. This visualization is designed solely through user interaction with iVisDesigner, without textual programming.

3.2 Design Philosophy

The framework of iVisDesigner is designed to represent visualizations that support interactive user manipulation, all within a web-based interface and canvas. The high-level design choices of iVisDesigner are based on the following idea: Allowing for interactive visualization creation, editing and interaction in an unified interface, where the space is dominated by a canvas showing the emerging visualization. We focus on enabling users to freely place graphical elements and links between them, instead of simply designing a chart or template visualization.

Our overall approach can be characterized as introducing support for data influx and manipulation to the common usage paradigm of interactive vector-based drawing software. By allowing users to define mappings from data to graphical elements, we enable them to directly create and manipulate groups of elements simultaneously, which tremendously reduces the amount of work to create visualizations. Transformed, aggregated, or otherwise generated data can be attached to the dataset, providing more capabilities, such as histograms and graph layouts. Graphical elements can be manipulated via dragging and brushing, which affects the underlying data attributes, enabling the design of interactive visualizations.

3.3 Design

In this section, we present the workflow and design details of iVisDesigner. We first give an overview of our framework, which is designed to represent visualizations, render them, and allow users to manipulate them interactively. We then discuss our user interface design, explaining how users can create and edit the different components.

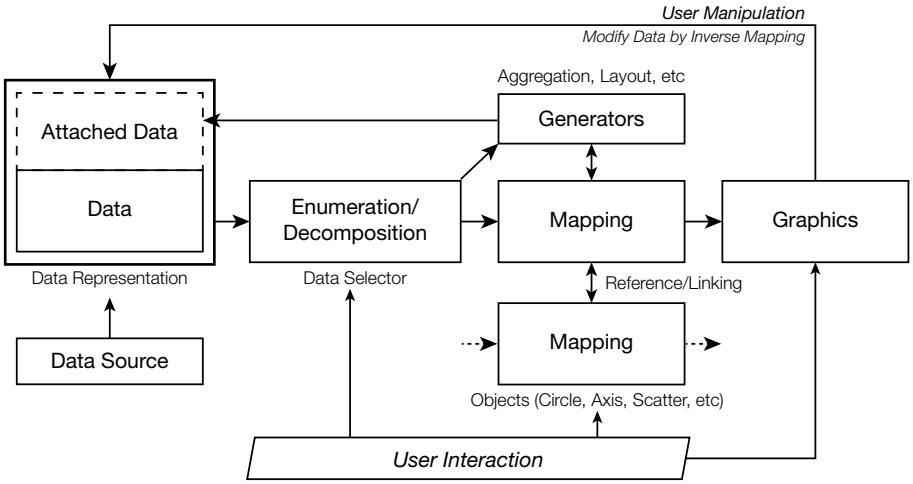


Figure 3.3: The framework of iVisDesigner. Data is first being enumerated by user-defined data selectors, and then passed to different mapping objects. These objects may refer to each other, and finally generate the graphics or attach generated data back to the data representation. Users can also manipulate the graphics, and modify the underlying data if permitted.

3.3.1 Framework

The framework of our system is illustrated in Figure 3.3. Data is loaded from the *Data Source*, transformed into an internal representation, and then enumerated or decomposed into individual elements by various *Data Selectors*. The decomposed elements are then passed into *Objects* for visual mapping or data generation. For visual mapping, the resulting graphical elements are simply rendered on the canvas. For data generation, the results are additional data attributes (*e.g.*, a histogram) that can be attached back to the data representation. In addition, users can create and attach new data from scratch, for example, creating a range of integers from 1 to 100 for numbering purposes. Multiple mappings and transformations can co-exist in the same visualization, and can refer to each other.

Our data representation is based on a hierarchical model similar to JSON, but allows for references among objects. Each data items is a set of key-value pairs. A value can be a single data item, an array of data items, a primitive value (number or string), or a

reference to another data item. The structure of the dataset is defined by a fixed *Schema*, which stores a definition of the data structure. This requires the items in a single array to be homogeneous, *i.e.*, they must be of the same type and structure. This ensures that we can perform mappings from each array of objects to graphical elements in a unified way. Given that the arrays in a given dataset are homogeneous, it is easy to construct a schema for a dataset automatically.

This dataset definition is relatively more expressive than tabular structures. For example, in Figure 3.2, the depicted dataset contains a set of air-probing stations, each with a set of measurements, which are visualized individually on the map, and collectively on the timeline plot. Users can place small visualization designs for inner-level data within a larger plot.

Users can create *attached data* by creating special objects in the system. These can, for example, compute basic statistics or run a force-directed layout algorithm.

Data Selectors, which are automatically created from user interaction, are used to select a set of data items or values from the dataset. While Data Selectors are created via the UI (*e.g.*, by clicking on an entry of the Schema panel or selecting from a dropdown menu in the Property panel), they also do have a syntax (string representation), where `[array]` means an enumeration of all the elements in the array, and `field` means a particular field from the current object. The Data Selectors can be specified in a path-like manner, joined by ":". For example, `[cars]:acceleration` means an enumeration of all (say n) cars in the dataset, taking the value of the acceleration field for each, resulting in an array of n numbers.

Reference fields can also be selected with the Data Selectors. For example, the selector `[edges]:&source` will select the source nodes for the edges in a graph (nodes are stored in a separate array), and `[edges]:&source:value` will select the `value` attributes of the source nodes.

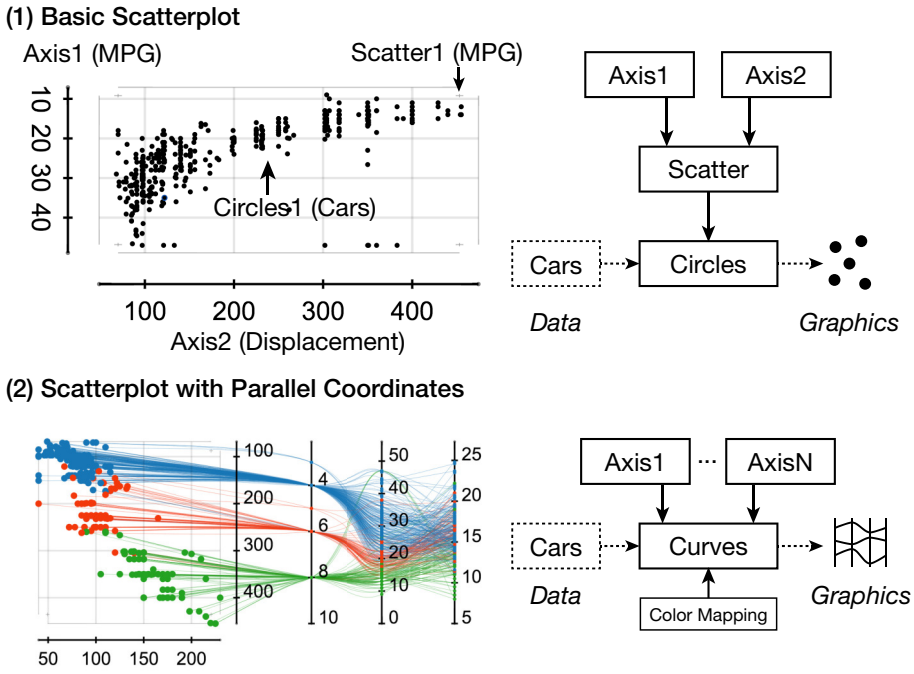


Figure 3.4: Two basic visualization designs: (1) Scatterplot (2) SPPC, as in [139]. The scatterplot consists of two *Axes*, a *Scatter* and a *Circles* object, with *Axes* and *Scatter* objects providing location information for the *Circles* object, which maps cars to circles.

A visualization consists of a set of *Objects*, which define mappings from data to graphical elements, or generate new data attributes and attach them back to the data representation (as shown in Figure 3.3, Mapping and Generators). Objects in our system can be of various types, specifically, they encompass *Graphical objects*, *Guide objects*, and *Generator objects* as discussed below.

A *Graphical object* represents a mapping from a set of data items to a set of graphical elements. Examples are *Circles*, *Lines*, *Polylines*, *Arcs*, and *LineThroughs*. Each Graphical object has a Data Selector associated that specifies the set of data items to map from, and each item in this set is rendered as a graphical element.

The properties of a Graphical object, such as the location and radius for the *Circles* or end points for the *Lines*, are provided by *Guide objects*. These objects transform data values to various visual properties such as location, width and color. Examples are *Axes*,

Scatters, and *Maps*.

Generator objects attach derived data to the dataset. For example, they can calculate the average value for a group of data attributes (*Statistics* object), group them into bins (*Aggregator* object), compute an expression on a set of data items (*Expression* object), or perform a force-directed layout algorithm on a graph and give each node a position (*ForceLayout* object). Generator objects mainly perform data transformations, they attach generated values to the dataset. Guide objects are displayed and edited on the canvas, and mainly deals with visual properties. There is no exclusive separation between the two categories, and one object might be of both kinds at the same time.

Generators can also accept user interaction. A *BrushingValue* object accepts brushing actions on a given visualization, and attaches corresponding data attributes to a data item that got brushed. These generated or derived values are attached to the dataset, which can then further be used in other parts of the visualization. For example, they can be used to construct brushing and linking functionality for a visualization, enabling basic visual analytics.

Objects can be nested into a *Component object*, which is bound to one Data Selector and which provides local coordinates for the objects inside it. Since our data representation allows for a hierarchical structure, users can design sub-visualizations and scatter them around. For example, one can create a component to design a small glyph for representing different data items. In Section 3.5, we show an example that uses components to create small timeline plots for each item (cf. Figure 3.2), and Figure 3.7 showcases some custom glyphs.

Note that object types are not mutually exclusive. One object might have several types simultaneously. For example, an *Axis* is at the same time a *Guide object* and a *Graphical object* showing tick markers. Force-directed layout is classified as a generator object, because it generates coordinates, but it can also be a guide object at the same time

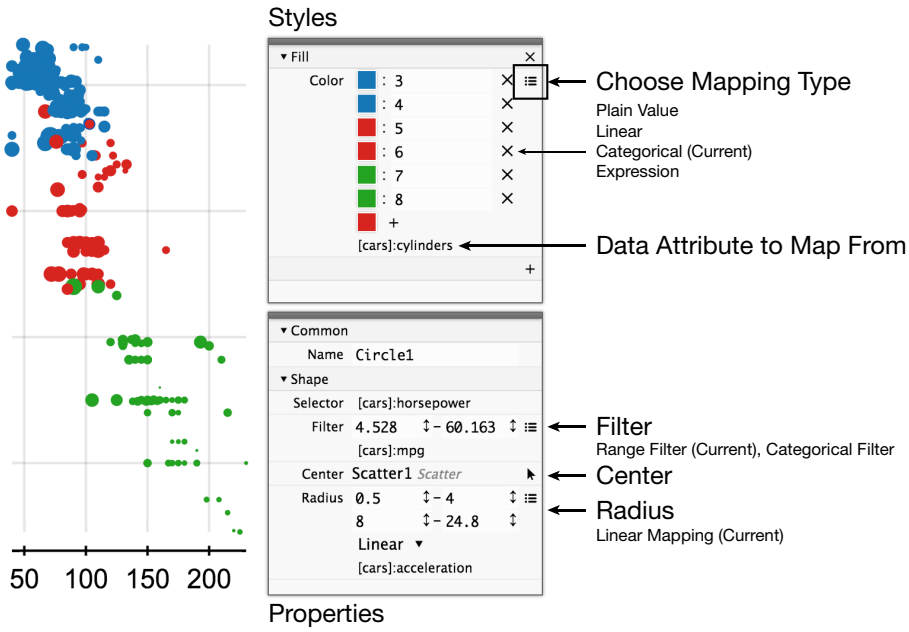


Figure 3.5: The properties and styles of the objects can be defined in the corresponding panels. Styles are drawing actions, such as Fill and Stroke; properties define the shape of the object, such as Center and Radius. Both, properties and styles can be defined as plain values or mapped from data via the UI.

(users indicate a rectangle for the layout region).

Our framework can be extended via programming by defining additional custom objects, ranging from primitive graphical elements to complex visual designs (*e.g.*, a special type of glyph).

A comprehensive list of currently supported objects and their properties, as well as the interactions to create each of them, can be found in the supplementary material.

3.3.2 Interface and Interaction

We now discuss the user interactions in iVisDesigner. The overall user interface is shown in Figure 3.2, together with an example design (detailed in Section 3.5.2). It consists of a menu bar, a set of panels, a status bar, and a drawing canvas. There are five flexible panels that users can freely move around, resize, minimize or hide: The

Tools panel presents a set of tools, including select/move, drag, brushing, pan/zoom, and object creating tools. The **Schema panel** shows the schema (hierarchical structure) of the dataset; users can select arrays or fields in the dataset. The **Object panel** shows a list of all graphical objects in the visualization and allows users to select, reorder or remove them. The **Style panel** is used to define graphical styles (series of drawing actions) for graphical objects. The **Property panel** lets users edit properties of selected objects. In addition, a data inspector panel is available, it can be shown when users need to examine the actual data values.

After an object is selected in the object panel or directly from the canvas, the property and style panels let the user edit the properties and drawing actions.

We aimed for a uniform interface, flexible for different tasks, and intended to maximize the space for the drawing canvas, resulting in the five panels above. The tools panel helps switch between different mouse tasks (select, interact, create). The schema panel and object panel are for selecting data elements and visualization objects, while the style panel and property panel allow users to edit the selected objects in an uniform way.

In the following sections, we will discuss typical steps to use our system, including actions to create a visualization, edit an existing design, and interact with them. Since the system is flexible, users could go back and forth among these steps at will.

Creating Visualizations

A visualization is created by adding graphical objects to the canvas. Typical steps to create a graphical object include: (1) Selecting a desired set of data items from the schema panel. (2) Selecting the desired type of object from the tools panel. (3) Providing initial positioning. Other properties of the newly created object will be set to default values, for later adjustment.

While in other systems, such as SageBrush, Tableau or Lyra, one directly assigns

data properties to marker properties, and the system will automatically determine scales and their positioning (that might be changed later), we require users to create guide objects (such as Axes, Scatters) explicitly. Since our canvas is virtually infinite, and there might be multiple existing visualization parts on it, automatically creating axes is not as straightforward. Therefore, to create a visualization, users first need to create guide objects as a frame, then add graphical elements. For example, to create a scatterplot, users need to first create two orthogonal axes, add a scatter between them, and then put, *e.g.*, circles on the scatter.

References in the dataset can be utilized. For example, in a node-link graph visualization, the edges are defined as two references to nodes, `source` and `target`. Suppose we already visualized the nodes as a scatterplot. We now want to create lines for the edges to make a node-link visualization. The lines are bound to the `[edges]` array, so each edge is drawn as a line. To specify the two end points for each edge, we need to use the locations provided by the nodes' scatterplot. In this case, the user first highlights the "ref" button next to the `source` reference field, indicating she/he is going to use the *node referenced by that field*, and clicks on the scatterplot to specify the first end point. Then the user highlights the "ref" button beside the `target` reference field, and clicks on the scatterplot to specify the second end point and lines are created. In short, the scatterplot defines a mapping from nodes to locations, and the reference field tells the scatterplot which node to use for the mapping. This is perhaps the most difficult-to-understand interaction in our system. While users in our evaluations were able to easily follow our guidance to create node-link graphs, it proved challenging for some of them to create other forms, such as the matrix and arc-based graph visualizations in Figure 3.9, given just a short amount of learning time.

To reduce the burden of complex interactions to create common designs like a scatterplot, we defined a small set of templates (scatterplot, timeline plot, node-link graph), that

allows users to create such visualizations simply by selecting the data attributes and dragging a rectangle on the canvas. The template will create the required graphical objects and guide objects for the user automatically, and the user can adjust the design later. This also allows novices to get started with the system more easily.

Editing Properties and Drawing Styles

After having created objects, users can further modify their properties. This is done via the property panel and the style panel. The property panel shows a grouped list of property-editing UI components. Most of the properties for an object can be set as mappings, such as linear mapping or categorical mapping, which are guide objects created implicitly by the property editor, allowing users to assign mappings from data attributes to actual properties of each graphical element represented by the object. For example, the "radius" property of the circles object can be assigned as a constant value, or as a linear mapping of some data attribute. The "center" property of the circles objects can be set by picking a guide object or a static point on the canvas. Properties can be copied and pasted among objects. Pasting can be done either by value or by reference. If pasted by reference, changing one of them will cause the others to change as well. For example, we might design a set of parallel coordinates and a scatterplot, and share the color mapping for the parallel coordinates and the scatterplot. Through copy and paste, one can reuse the properties. However, one drawback is that there is no easy way to indicate what objects share the same mapping in the interface. Currently, the user has to remember that. Another limitation is that our current version does not provide a straightforward way to visualize the scales of the mappings. In future versions, we would like to allow users to drag the mapping properties out onto the canvas and draw them as interactive scales. This would solve both of the above problems.

The style panel works similar to the property panel, showing a list of drawing ac-

tions for a graphical object. In the rendering process, each graphical object generates a graphical path, for example, lines, circles, Bezier curves or composites of these, and this path is rendered to the canvas by performing the drawing actions specified here. For example, the "Stroke" action will stroke the path, and it has four properties: width, color, line join and line cap. The "Fill" action will fill the path with a user-defined color. Users can add/remove actions, and reorder them in the style pane. The properties for drawing actions in the style panel can also be mappings, as in the property panel. The user can add or remove these actions in the style panel. Styles can be considered a special set of properties for a graphical object frequently used in visualization designs, and they commonly consists of multiple actions. This is the main reason we separated them from the property panel into an individual panel. Currently we only have stroke and fill actions, in the future, we would like to support more actions, including such that alter the path (*e.g.*, distortion, outline, smoothing), similar to those in Adobe Illustrator.

Basically, we employ a property-editor based approach, which provides a set of uniform editing steps for each type of properties. The editing interfaces are automatically generated according to the properties *declared* in the object types, which is particularly useful for implementing new types of objects. From the users' perspective, a uniform editing experience helps them to learn the system effectively and speedily.

Interacting with Visualizations

In addition to creating and editing visualizations, our system allows a certain set of interactions to be designed. There are two specific tools for interactions on the visualization design.

Moving elements Users can move graphical elements in a designed visualization, and as a consequence, some corresponding data attributes might be changed. This should

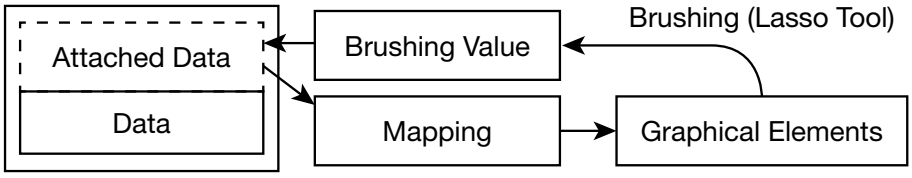


Figure 3.6: The internal process of brushing. When the user brushes over a set of graphical elements with the lasso tool, they are collected and passed to the BrushingValue object (which can be created and activated by the user). The BrushingValue object then attaches data back to the data representation, and the attached data can be used by mapping objects, which affect the graphics. Users can define and combine multiple ways of brushing in our system.

be performed very carefully, as one could easily produce fake findings, mislead others or get confused when changing the original data recklessly. As a default setting, we don't allow the original data attributes to be changed. However, certain properties can be changed without danger, such as the locations produced by a force-directed layout algorithm. Another example consists of users attaching a single attribute to the dataset, and using an axis and a circle to build a "slider" to control it. Such slider-controlled attributes could be used in different situations, such as, *e.g.*, defining the range of a filter property, resulting in an adjustable filter.

Brushing The lasso brushing tool is paired with the *BrushingValue* object, which attaches a value to each data item, and once data items are brushed by selecting their graphical elements, their values will change. Users can then use the values to define graphical mappings. This implements brushing and linking functionalities. For example, one could set the fill color of a scatterplot as the brushed value, and use the lasso tool to color the points. Other examples are discussed in Section 3.5.

Our goal is to enable the design of interactions, not just supporting a fixed set of interactions. This is achieved by allowing users to modify attached data attributes in two classical ways (moving and brushing), and we show that dragging, filtering, and brushing and linking can be supported from these atomic actions. There are possibilities to define

more complex interactions beyond these, but as the complexity goes up, users have to create several related objects such as axes and expressions, which makes the process a little more complicated.

3.3.3 Limitations

While the framework of our system is inherently modular and object-oriented, allowing new types of objects to be added easily, it still has several limitations in terms of expressiveness. The framework is based on designing and parameterizing graphical mappings from original, transformed, or user-generated data. This approach has two fundamental limitations. (1) Designing adaptive markers, such as automatically determining the width of bars in a barchart based on the number of bars and the chart width, requires writing specific mathematical expressions, because this involves dividing the chart width by the number of bars, which is not a property of any single data item. In general, our framework does not address the dependencies among graphical objects, but rather performs mappings individually, so in order to accommodate higher-level layout constraints such as overlap avoidance, special objects have to be designed in programming. (2) Our system cannot, without using specific custom layout objects, design recursive drawings such as tree maps, and as aforementioned, the system doesn't support recursively defined data structures directly. One might argue that tabular structure can represent graphs and trees as well, but our data selectors can only enumerate arrays of items in the data hierarchy. It cannot follow references (such as running a graph/tree traversal). This is also true for other declarative approaches (*e.g.*, ProtoVis, D3, Vega and Lyra), which are also resolved to using specific modules for each kind of layout. Abstractions such as [140] might be considered in the future. These limitations currently exclude a range of possible visualization designs.

There are also shortcomings that are more easily solvable within the current framework. (1) Our system currently is constrained in terms of the type of coordinate systems. Positional mappings are done via axes and scatters (the map with Mercator projection is the only exception). It does not currently accommodate circle-based visualizations, nor are polar coordinates currently supported. In the future, we will seek to support different coordinate systems. (2) Axes and their scales are currently intrinsically linked. For simple scatterplots and parallel coordinates this works fine, but it becomes tedious, although not impossible, to share the same scale for different data attributes. There is currently no way to use axes for numerical properties such as widths or radii (and this would be useful for designs such as error bars). This could be solved by additional property-editing interactions and better separation of axes and scales in the future. (3) There is currently no way to specify the order of data item enumeration, which would be useful if we want to stack data items or link through them in different manners. This could be solved by adding a sorting attribute to the data selectors. (4) The system currently lacks a way to specify more general graphical paths. The *LineThrough* object can draw paths through data points, which is somewhat restrictive: if we want to fill the area below a timeline plot, or between two of them, we need a more flexible way to define paths to connect static points and sequences of points together (similar to the "Pen" tool in Adobe Illustrator). The consequence is that our system is less expressive at defining shapes, and more oriented towards line-based visualizations. A general "pen" tool would be desirable.

Despite these limitations, our system can still support an extensive set of visualizations. In Section 3.5, we showcase a variety of examples.

3.4 Implementation

The system is implemented in HTML5 using jQuery and other open-source libraries. A backend server written in Python Django is used to store the metadata for the datasets and saved visualizations. Here, we discuss some notable aspects of the implementation.

Input format The datasets are loaded as JSON objects, where the references are stored as the referenced items' ID (each data item having a unique ID). For our current system, we did not focus on supporting multiple data formats, but conversions from CSV or Excel-like data sources are trivial (without performing join operations). Dataset structure will influence how well certain designs will be supported. For example, if the dataset from Section 3.5.2 had been stored as a flat table instead of a hierarchy, we would not be able to create that visualization, unless we added a special "Grouping" generator object, similar to Lyra's approach).

Rendering We employ multiple layers of HTML5 Canvas. The renderer maintains the status of these canvases, and executes the visualizations on them. There are four layers in our current prototype. The Main layer contains the graphical elements, the Front layer shows selected elements, the Overlay layer shows temporary markers, such as alignment indicators during user interaction, and the Back layer displays the background color and grid. By using layers, we eliminated the need to redraw the entire visualization when the user selects a single element, achieving better responsiveness. In addition, the renderer also manages a viewport, allowing users to move or zoom the visualization, or to export the current view as PNG or SVG files.

Serialization The visualization is stored as a set of JavaScript objects internally; to save a visualization, we need to serialize them to a storable format. We implemented a

general JavaScript object serializer to support this task, which is capable of maintaining references between objects and retaining type information, which are critical for correctly restoring a visualization. To enable this, we assign a unique identifier (UUID) for each object, and store object references as UUIDs. Type information is preserved by recording an identifier for each registered object type, and restoring the "constructor" attribute for each object. The benefit is that we do not need to write a pair of serializing and deserializing functions for each object class, which reduces programming effort and the possibility of bugs.

Backend server The system operates mainly in the browser, but like every web application, it requires a backend server to provide data and store information. The backend server manages user accounts, datasets and visualization designs. It was implemented in Python Django and Twisted. The Django part is responsible for managing user credentials, storing the metadata of all datasets, and saving and loading of visualization designs. The metadata of datasets consists of the data description, data schema, and a URL for the data content. We used the WAMP protocol (based on WebSockets) in a Twisted server, which is connected to a Redis database. It provides real-time updates for changing datasets. Changes in the dataset can be posted to the web-based interface, and the system will update the visualization with the changed data. One can also write scripts that collect data from the web, and send it to iVisDesigner (either replacing original or providing incremental updates). For example, Figure 3.1 (3) illustrates real-time monitoring of a server's CPU, RAM, and network usage.

Embedding Users can export their designs and embed them into their own websites or web applications. The datasets and visualization designs could either be retrieved from a server, or embedded statically.

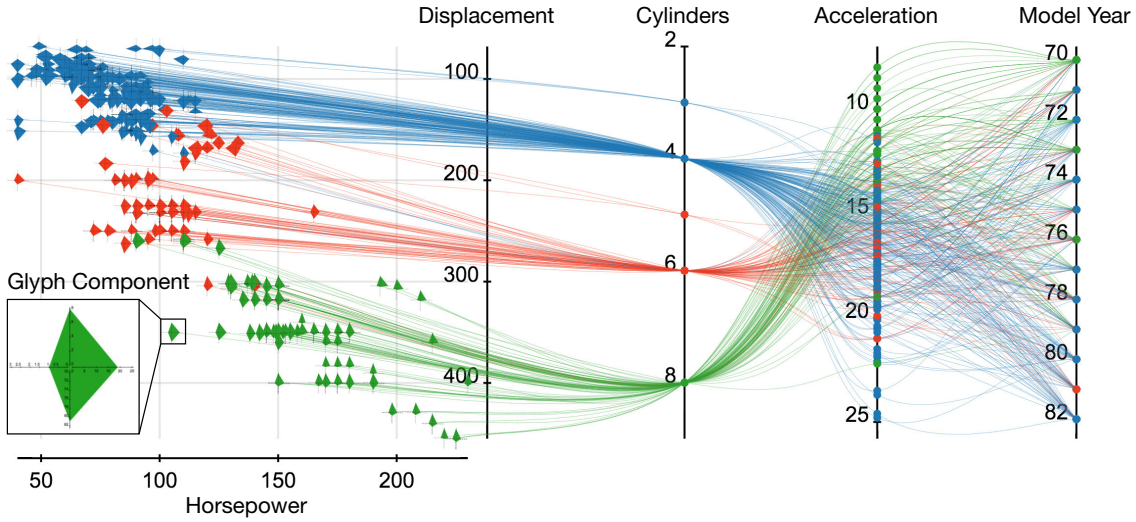


Figure 3.7: Glyph-embedded Multidimensional Data Visualization. This design is based on the SPPC design [139], connecting a scatterplot with parallel coordinates. The points in the scatterplot are replaced by a set of glyphs, showing four attributes for each item.

3.5 Example Applications

In this section, we present a set of visualization design examples on different datasets, with the goal of illustrating the flexibility and expressiveness of our system. Different types of datasets are chosen, including multidimensional data, time series data, and graph data. We also demonstrate a design for Sina Weibo (A Chinese microblog service similar to Twitter) user data, and even some artistic designs without an underlying dataset.

3.5.1 Multidimensional Data

iVisDesigner can flexibly arrange axes like Flexible Linked Axes [62], whose design space is subsumed by our system. Figure 3.4 (2) is an example of linked-axes and scatterplot-based visualization designs. The dataset used there is the 1983 ASA Data Exposition Cars dataset [141]. iVisDesigner emphasizes expressiveness. For example, we can design mini-glyphs for Cars with the *component* object, draw the glyphs on a scatterplot, and link them to a set of parallel coordinates (Figure 3.7).

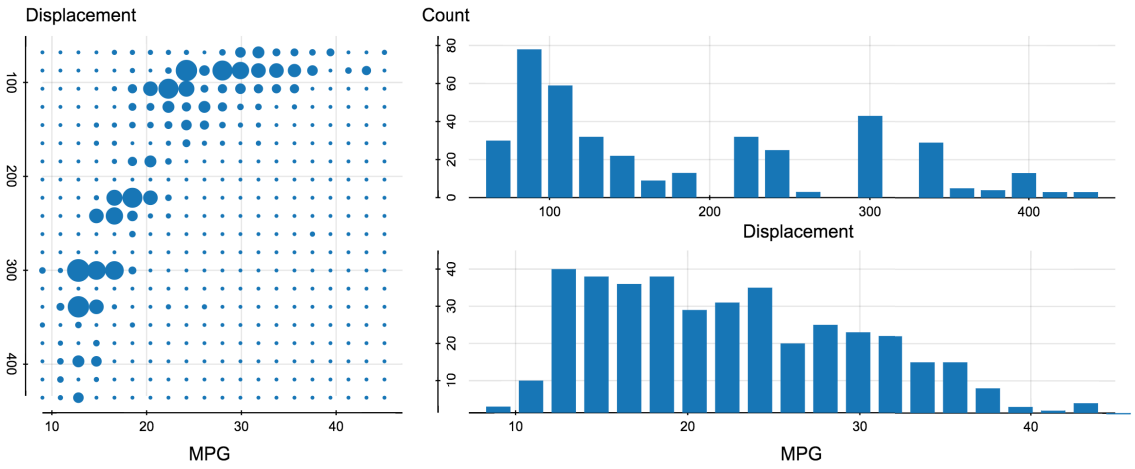


Figure 3.8: *Aggregator* objects group values into numerical fields, which can be used to draw a histogram of a particular data attribute. In this example, we plotted the histograms of MPG and Displacement attributes in the Car Data, and also a 2D histogram to show their joint distribution.

Generator objects can be used to compute the statistics of a data attribute, including basic statistics such as min/max, mean, and more involved ones such as histogram. In Figure 3.8, we show an example using the histogram generator, an *Aggregator* object. This object generates an array of bins to form a histogram of the selected data attribute, which can be displayed in various ways.

3.5.2 Time Series Data

In Figure 3.2, we presented a visualization of the Beijing Air Pollution Dataset, crawled from two websites that update hourly. The dataset contains 36 stations, each of which has a name, a geographical location, and a time series of two weeks of measurements. The visualization consists of two different charts. The left chart is a set of timeline plots on a map, each representing a station's measurements. It shows the measurements for each station, allowing comparison between stations at different locations. The right chart is a single timeline plot, which contains the timelines for all of the stations. This chart shows the main trend of all the stations, while revealing some outliers. The left-hand

visualization consists of a *Map* object for the geographical coordinates, two *Axis* objects, a *Scatter* object and a *LineThrough* object, connecting all the points in sequence.

3.5.3 Graph Data

A graph visualization with both, node-link diagram and adjacency matrix representations, is presented in Figure 3.9. The graph is based on character co-occurrence in Hugo's *Les Misérables*[1]. The dataset contains a set of nodes and a set of edges, each edge referencing source and target nodes. The node-link diagram is constructed by first creating a *ForceLayout* object, which runs the Fruchterman-Reingold algorithm [142] to compute the layout, and then attaching the resulting coordinates (x and y values) to the nodes. The nodes are then drawn as a scatterplot of the attached x and y values. The edges between nodes are drawn using references to the node scatterplot.

The matrix representation is created by first assigning an index for each node by the Expression object, then the edges are scattered as *Circles* with the source node's index as the x axis, and the target node's index as the y axis. Since the index attached by the *Expression* object can be changed, users can use the "MoveElement" tool to drag the labels on the left of the matrix to re-order the nodes.

This visualization design also supports brushing and linking. We added a *BrushingValue* object for the edges, so users can select a set of edges in the node-link diagram or in the matrix, and get them highlighted in both views. Since the *BrushingValue* object supports brushing both numbers and colors, we can color the edges or change their widths by brushing. What graphical attribute (color, width) to brush is up to the user. With our system, users are free to design their way of brushing and linking, and have end-users perform it interactively.

[1]Dataset compiled by Donald Knuth, retrieved from `http://bl.ocks.org/mbostock/4062045`

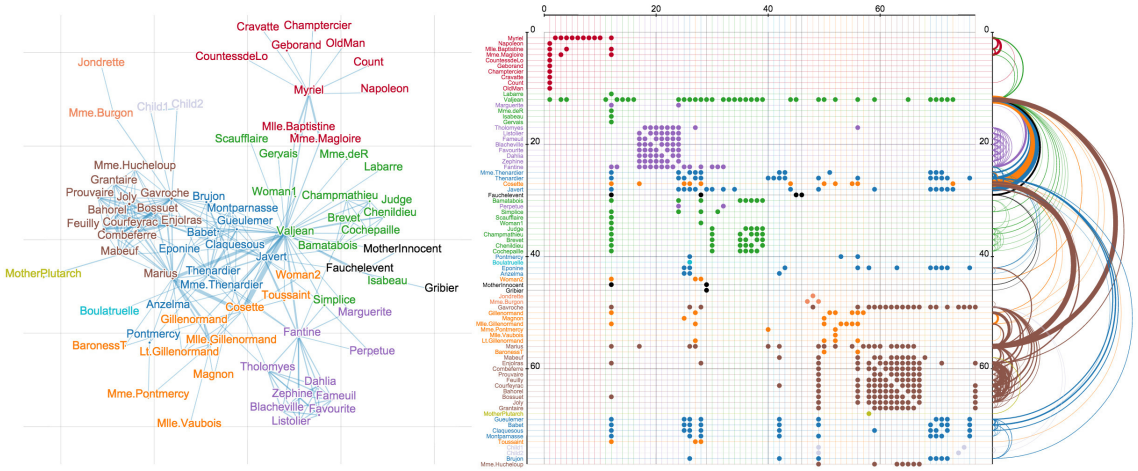


Figure 3.9: Visualization design for the co-occurrence network graph from *Les Misérables*. Left: Node-link diagram with force directed layout. Right: adjacency matrix design. A brushing and linking mechanism for the graph edges is designed into this visualization. When the user selects a set of edges from either the left view or the right view, these edges will be highlighted in both views.

3.5.4 Social Network Data

Next, we present an example with flexible linking between different views. We set up a data connection with WeiboEvents [143], which crawls data from Sina Weibo for iVisDesigner. The user can enter an account name in Weibo, then the crawler will crawl the account's tweets, friends and followers, and send the resulting dataset to iVisDesigner, where users can create visualization designs. Figure 3.10 shows a visualization designed for such a dataset. It consists of a map showing the account's trajectory by connecting all of its geo-tagged tweets on the map. The map is linked with a time axis, revealing where the user was located during each time period. The right side contains several scatterplots, showing the statistics of the account's followers. Users could employ our system to create and connect various components, for example, they could move the time axis around, and see the connections more clearly, or link the map to the bottom timeline.

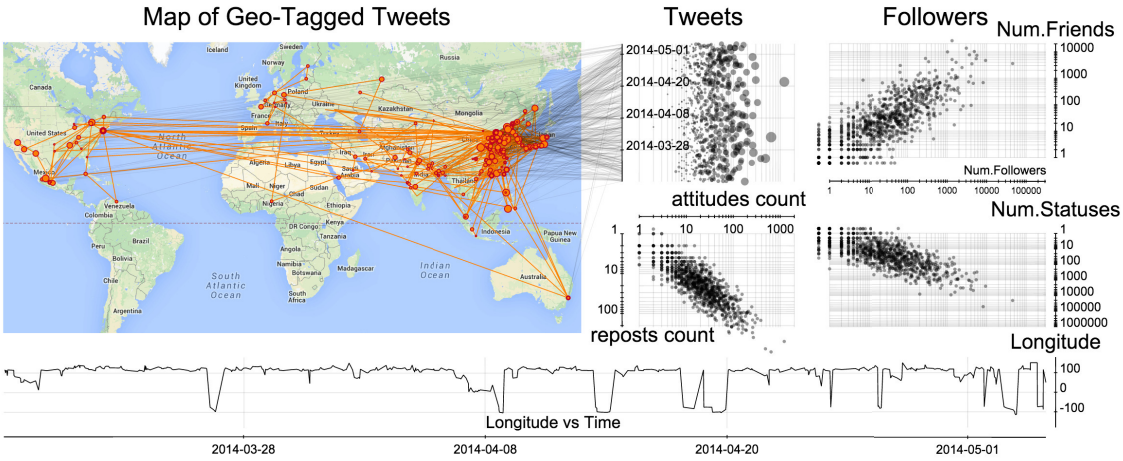


Figure 3.10: Weibo User Visualization. The datasets were crawled from Sina Weibo, including the metadata of the recent tweets of a selected user, their followers and friends. This visualization shows the user's trajectory in a map view, and links the map view with the time axis. Scatterplots of four joint distributions are shown. Due to privacy concerns, the data shown here is synthetic, roughly modeled on observed distributions only for illustrating the visualization design. Readers should not draw any conclusion about Weibo users from this visualization.

3.5.5 Generating Data from Scratch

In iVisDesigner, users can also create graphical designs without using an underlying dataset. This can be useful, *e.g.*, to illustrate some mathematical concepts. In Figure 3.1 (6), we plotted the Butterfly Curve [144] on the canvas. This is done by first creating a *Range* generator object, which creates a range of numbers from 0 to 24π. Then, we created two *Expression* objects, each of which takes the generated numbers in the range, computes the parametric expressions for the x and y coordinates, and attaches the values to the *Range's* items. Finally, we created two *Axis* objects, a *Scatter* and a *LineThrough* object to visualize the function. The radii of the *Circles* are also bound to the function value in this case. After initial setup, users can interactively change the *Range* and the *Expression* as well, or define other mappings such as the color of the *Circles*.

In Figure 3.11, we created an interactive illustration of Bézier curves. We first created a *Range* of numbers from 0 to 1 as the t parameter in the curve, then created four *Circles*

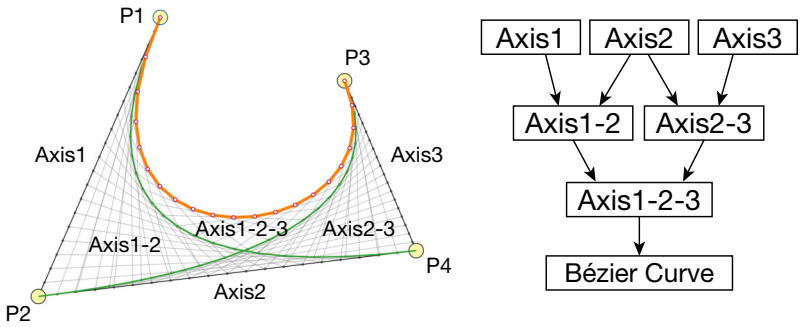


Figure 3.11: Interactive Bézier-curve illustration: users can drag the four control points to change the shape of the curve.

as the control points. Next we created three *Axis* objects connecting the four control points, then two *Axis* objects between the previous three, one *Axis* between the previous two, and finally the *LineThrough* that connects all the points in the Bézier curve. After the configuration, users can move the control points freely, the curve will change according to the user's interaction.

For these examples, we didn't employ any dataset. In our system, users can not only create data visualizations, but also create mathematical illustrations or even artistic designs, using a set of graphical objects and generator objects.

3.6 Evaluation

In this section, we first present a performance evaluation of our system and then show the results of an informal user study we conducted.

3.6.1 Performance Evaluation

We analyzed the overhead added by iVisDesigner's mechanism for rendering visualizations. Our experiments were done on a MacBook Pro with 2.6GHz Intel Core i7 processor, 8GB RAM, running MacOS X 10.9.2. The browser used was Google Chrome

33.0.1750.146. We compared the rendering performance of our system with hard-coded JavaScript and D3.js.

We created three visualization designs for the test: (1) Scatterplot with uniform size: a scatterplot for the Cars dataset (406 cars), showing MPG and Displacement, with circle size 5. (2) Scatterplot with mapped size: the same scatterplot, but the circle radius is mapped as the number of cylinders. (3) Timeline: A timeline plot showing minimum temperature in a particular place over 115 days. The times to render these visualizations are 7.3ms, 7.6ms, 0.3ms respectively. The hardcoded version runs 5 to 7 times faster (1.7ms, 1.5ms, 0.04ms) than our system. This is because we have an extra layer of data enumeration and mapping, which involves a lot of function calls in the code. D3.js is around 6 to 30 times slower (61.3ms, 52.2ms, 11.1ms) than our system. Since our system uses HTML5 Canvas as the rendering engine, it is not a fair comparison with D3.js, which renders graphical elements as SVG elements, but since D3.js is a successful programming-based visualization framework that is seen as reasonably efficient, we see this result as encouraging. We chose Canvas as the rendering engine because it is very fast, and we do not rely on the simplified mouse events provided by SVG, since our system itself is responsible for handling mouse events.

We also measured the amount of time to render the visualizations in our design examples. The Graph in Figure 3.9 with 77 nodes and 254 edges, takes 43.54ms to render, the SPPC in Figure 3.4 (2) with 406 cars takes 28.61ms, and the Beijing Air Pollution visualization in Figure 3.2 with 3132 measurements for 36 stations takes 90.49ms to render.

The results of the performance evaluation shows that our prototype system is able to handle visualizations with hundreds or a few thousands of graphical items at real-time or at least interactive frame rates. The performance can be further improved by incorporating optimization techniques, such as reducing the number of function calls

and auxiliary objects in the rendering process.

3.6.2 Informal User Study

We conducted an informal user study for our system. The user study was designed to solicit feedback from real users of the system, after showing and teaching them the basic principles. We recruited 8 users, 4 male and 4 female, ranging in age from 24 to 32 years, with a median of 25, most of whom had Computer Science backgrounds and were familiar with computer-based visualization concepts. We also recruited one of them for a supervised study in the style of a thinking-out-loud cognitive walkthrough.

The informal user study was conducted on an online web interface, with participants performing various tasks, and optionally asking the supervisor questions. After a short introduction of the iVisDesigner tool and the goal of the study, users watched an 8-minute video tutorial (similar to the supplementary video) explaining basic steps in creating a scatterplot for the Cars dataset, a graph visualization of the character co-occurrence data, and some brushing and linking interactions on existing designs. Next, they were asked to try the system by following some steps from the video to get an initial sense of the logic and interactions of the system, and then to try and create their own designs from what they have learned. Finally, the users were asked to complete a survey with Likert-scale (2 = Strongly Agree, 1, 0, -1, -2 = Strongly Disagree) questions. All in all, users spent on the order of an hour on the user study. These are the average results for the questionnaires: "iVisDesigner is {expressive (1.75), easy to use (0.63), easy to understand (1.13), useful (1.88) }", "iVisDesigner is good for { basic visualizations (1.75), novel visualizations (1.25), multidimensional data (1.75), graph data (1.50), time-series data (1.63), visual analytics (1.00), overview (1.75), artistic designs (1.00) }". Most of the ratings are towards the top of the scale (2 or 1), with a few lower scores on "easy to use" and "easy

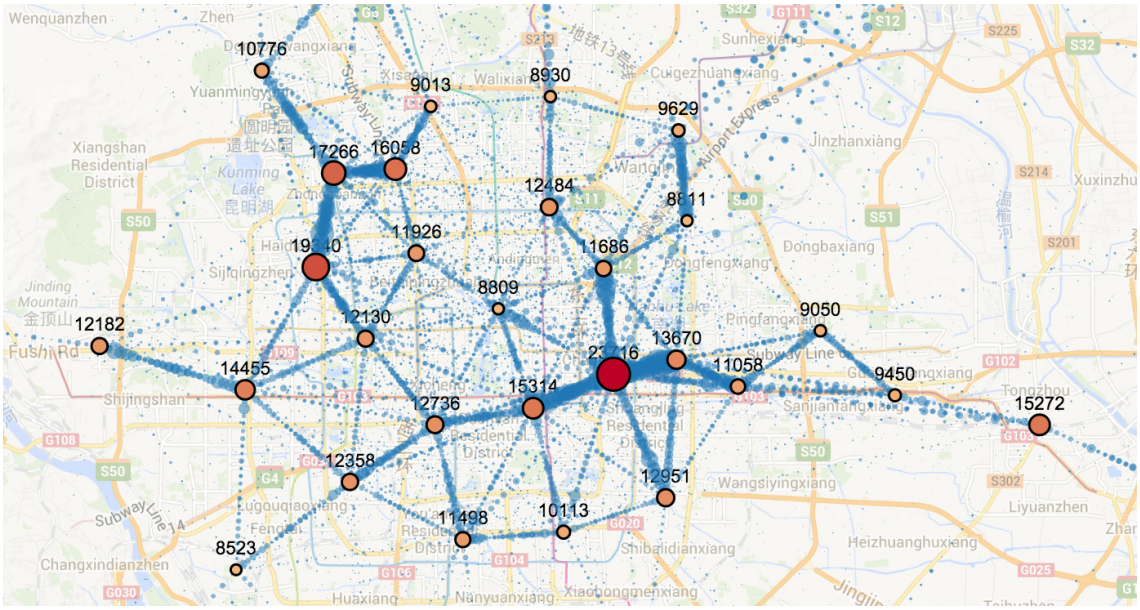


Figure 3.12: A visualization design by the participant from the supervised study. Each circle shows a cluster of tweets, dots between circles show the time-dependent movement pattern between clusters.

to understand". Participants believed that our system is very expressive and useful, and good for designing visualizations for different types of data. One participant said that the system is very flexible, he could *"make combinatory data/feature selection, for example, linking 2D and 1D elements together to create a polyline"*. From their textual comments, we observed that participants made use of templates very well; basic visualization designs could be created without difficulty. Ease of use and ease of understanding is more of a challenge for our system; as the participants pointed out, they had to carefully watch the video to learn the interactions. They also requested a more comprehensive user guide and tutorial. Usability could be improved by fine-tuning the user interactions. For example, one participant suggested we could enhance the highlighting when users select an element, making it more visible for better guidance. The tool-tip text in the status bar was also recommended to be moved upwards to attract more attention.

In addition, we recruited one participant, a researcher analyzing Twitter feeds for

social feature extraction, for a supervised study. We presented three datasets for the participant to explore. The participant was asked to construct visualization designs and try to understand the datasets though these visualizations. During this process, the participant could ask for help on how to use the toolkit and he also commented on his mental processes and considerations.

The supervised study was informative and successful in the sense of user appreciation for the expressiveness, speed, and stability of the system. The participant experimented with the Weibo user dataset (discussed in Section 3.5.4). Scatterplots were created to show the correlation of different attributes of a user's friends, and the participant made use of the brushing feature between two plots. Also, he tried to bind the radius of the points to the number of bidirectional followers. *"I could easily discover bot / celebrity clusters. Also, by varying the radius of each circle proportional to the number of bidirectional followers, I was able to locate commercially used / institutional accounts. I was also able to locate individual / organizational accounts by looking at the logarithmic scatterplots by having different types of attributes of users. The labeling is more flexible than other visualization frameworks."*. The participant also experimented with the Weibo geographical dataset, which contains a set of users, each having authored a series of geo-tagged tweets. Before the experiment, the locations in the dataset were clustered by a K-means algorithm, and the trajectories of users were grouped as edges between clusters. The user produced the visualization shown in Figure 3.12. *"I'm able to identify usage of Weibo on each geographical area in the city. By applying timeline stamps on each edge between adjacent nodes (clusters), we can track the user movements between different locations over time. It is very interesting to see significant amount of communication and movement between adjacent nodes which can perhaps be the reflection of the physical proximity between the users." "I would have spent 1–2 hours to create this visualization by programming, it was done in a few minutes using iVisDesigner."* The user also tried our system on one of his own Twitter datasets for a timeline plot,

and identified some previously undiscovered date-time conversion problems in his data preprocessing. In summary, with some supervision, the user gradually understood the general process to create visualization designs in our system, and was able to apply his knowledge to create visualization designs and understand the datasets.

3.7 Discussion

We have presented iVisDesigner, a novel expressive interactive information visualization construction toolkit. iVisDesigner is able to cover a wider spectrum of possible visualization designs than previous interactive (non-programming) toolkits. We already discussed its limitations in terms of expressiveness in Section 3.3. Here we discuss usability concerns and possible future improvements.

As we increase expressiveness, the interactions to build a visualization design become more complex than required by more single-purpose toolkits such as [62], because we need to allow specification of extra design parameters, which other toolkits predefine. Compared to, *e.g.*, the Flexible Linked Axes work, we need to specify what data to map from and what types of graphical elements to use, in addition to the axes and scatterplots. There is clearly a tradeoff between expressiveness and complexity. A simple way to improve user accessibility is to add more templates for existing designs. Users could start with a common template, and then modify it to satisfy their own needs. During the design and evaluation of the system, we have observed that designing from scratch is much more involved than modifying an existing design; providing templates certainly helps lower the barrier to entry. Another possible direction is to automate some design decisions by trying to predict what the user may want to show, filling in suggested informed default values for more complex parameters.

The learning curve of our system is not low. One contributing factor is that we haven't

yet optimized online help and error reporting, but we are steadily improving on that front. However, when users have to learn a whole set of new concepts, such as axes, references and components, it will inevitably take some time for them to embrace the possibilities and fully utilize their potential for creative designs.

In the framework of iVisDesigner, we did not yet fully consider the interaction among graphical elements. For example, when drawing a graph, there might be multiple edges between two nodes, depending on the dataset. In this case, users might want to define some rules other than just placing two lines in the same place, for example, double the thickness, or use a different color. These types of designs are not feasible in our current framework (also not in D3.js). We could insert a new step in the pipeline of our system, after the mapping stage. Once we have all the graphical elements, we can allow users to define interactions among graphical items before they get rendered.

Dynamic visualization design is another future direction. Up to now, we have dealt predominantly with static visualizations, with the exception that we are able to re-render the visualization when the dataset is changed. However, users cannot define how a graphical element appears or disappears when a corresponding data item is added or removed. This could be achieved by adding a property on graphical elements that would let users specify various types of transitions.

3.8 Conclusion

In this chapter, we have presented iVisDesigner, an expressive interactive web-based information visualization construction toolkit. Our system was designed to be a flexible tool for interactively creating information visualizations, inspired by interactive vector-based drawing tools and established information visualization principles. The system allows users to freely place graphical elements, and links between them, on a

large central canvas. We chose a declarative approach to avoid reliance on familiarity with programming and for keeping the usage simple and straightforward. Our unified editing interface allows users to create and edit graphical, guide, and generator objects, enabling the interactive design of complex visualizations. We presented example applications to illustrate the breadth of design possibilities, discussed the limitations and future improvement possibilities of our approach, and reported the results of a performance evaluation and an informal user study. The source code of iVisDesigner is available at `https://github.com/donghaoren/iVisDesigner`.

Chapter 4

ChartAccent: Authoring Annotation for Data-Driven Storytelling

4.1 Introduction

In the previous chapter we have presented iVisDesigner, an interactive visualization authoring system. While iVisDesigner supports a variety of visualization designs, it does not support annotation which is an essential part of visual data-driven storytelling [20]. *The New York Times* graphics editor Amanda Cox once stated that *"the annotation layer is the most important thing we do... otherwise it's a case of here it is, you go figure it out"* [145]. For example, annotations in an interactive slideshow convey a narrative, providing explanations that viewers would be unlikely to identify on their own [4]. In addition to helping presenters explain core messages or specific data, annotations enable them to emphasize and draw viewers' attention to specific parts of the chart [63]. Furthermore, appropri-

The contents of this chapter have been previously published in *Proceedings of IEEE Pacific Visualization Symposium (PacificVis)*.

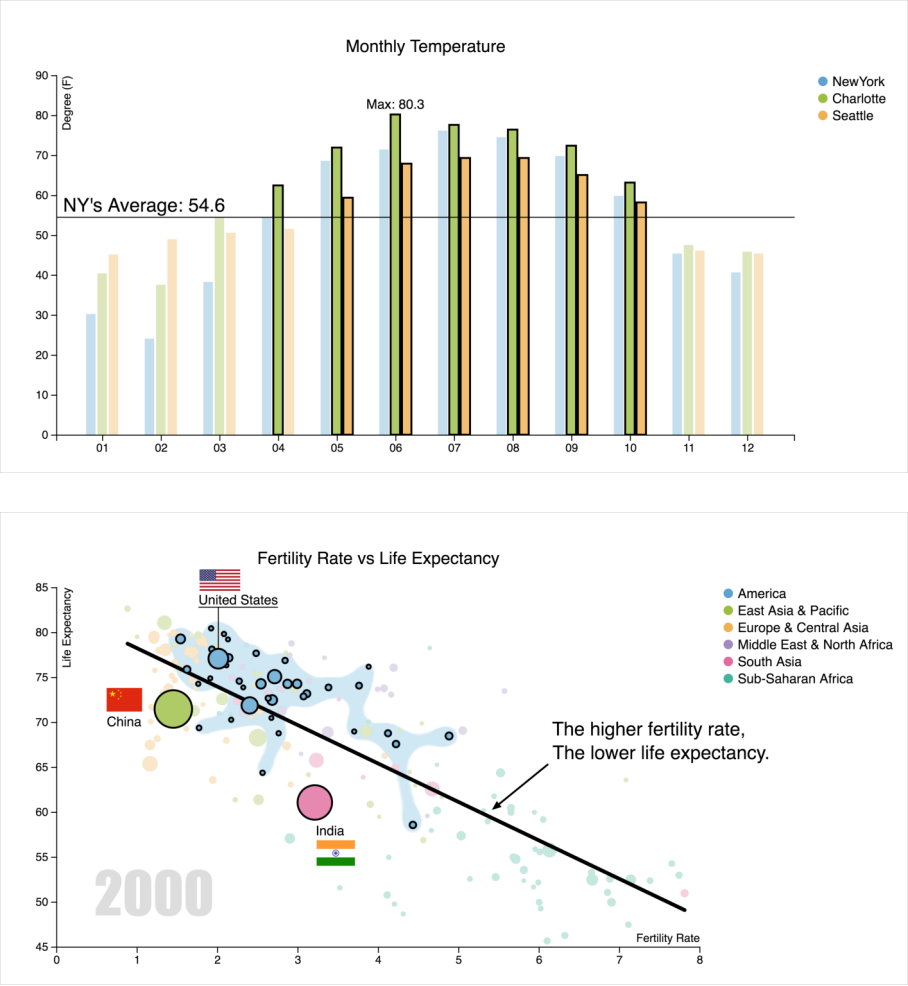


Figure 4.1: Examples of annotated charts created with ChartAccent: (top) emphasizing the months when Charlotte and Seattle's temperatures are higher than New York's average; (bottom) the relationship between fertility rate and life expectancy, with text and image annotations for the United States, China, and India; countries from North and South America are highlighted in blue.

ate annotations can help presenters provide additional context, potentially facilitate the memorability of a chart [146], and increase the aesthetic appeal of a chart.

However, commercial charting software such as Excel or Tableau provides limited support for creating annotations; similarly, current business intelligence tools that incorporate visualization provide little annotation support beyond simple text annotations [72]. Thus, to create an envisioned set of annotations, people often export charts to presentation authoring tools or graphic illustration tools [81, 82], where they can add individual annotations to a chart, albeit at the cost of being disconnected from the underlying data. For example, personal data presentations by "quantified selfers" often contain annotated charts to make key points more salient, interpretable, and enjoyable [22], but these annotations were often added manually. Considering its high importance and utility in storytelling, we argue that the visualization community requires a better understanding of the annotation design space: the forms of annotation, what purposes these forms serve, and how these forms of annotation can be applied to elements in a chart via user-driven and data-driven approaches.

In this chapter, we introduce ways to help presenters quickly and easily augment their charts with various annotations. We characterize a design space of chart annotations, drawing from a survey of 106 annotated charts published by six prominent news graphics desks including *The Economist* and *The New York Times*. We also discuss the reasons and goals of annotating charts, and what does and does not constitute annotation in this context. Informed by the survey and design space, we designed and developed ChartAccent, a proof-of-concept tool that provides a palette of annotation interactions that generate manual and data-driven annotations. We report on a reproduction study, in which we aimed to evaluate the experience of annotating charts with ChartAccent. After a short tutorial and practice session, most participants could easily annotate line graphs, bar charts, and scatterplots with ChartAccent, reproducing a series of previously

annotated charts. Furthermore, they enjoyed creating annotation with ChartAccent, and expressed a strong desire to use ChartAccent for future presentations. We conclude with a discussion on the lessons learned from the design and evaluation of ChartAccent, and we indicate future research directions.

The contribution of this chapter is threefold: (i) a reflection on chart annotation in the context of data-driven storytelling; (ii) design dimensions for chart annotation based on a survey of 106 annotated charts published by prominent news graphics desks; and (iii) the design, development, and evaluation of ChartAccent, which is now available for use at `https://chartaccent.github.io`.

4.2 Annotation Design Space

Existing definitions of "annotation" and the act of "annotating" relate to the altering of an existing object by adding a note or comment to it. Unfortunately, there is currently limited treatment of annotation in the visualization research literature, either in the context of design spaces [19] or in task or interaction taxonomies [147, 73, 148]. Thus we set out to survey a corpus of annotated charts and identify a design space for annotation.

With regards to the scope of our design space, there are several chart elements that we do not consider to be forms of annotation. First, we exclude graphical marks that correspond directly to the underlying data, such as bars in a bar chart or the points in a scatterplot. Unlike Kirk's characterization of annotation [145], we also exclude nameable graphical and textual elements associated with a variety of charts, maps, and plots; these include chart titles, legends, axes, axis labels and tick marks, as well as grid lines or graticules. Though we do not consider these elements to be annotations, they can certainly *be annotated*, as we explain below in Section 4.2.2. Finally, we constrain our scope to visual channels of communication.

4.2.1 Survey of Annotated Charts

There is an abundance of annotated visualization artefacts that could help us identify a design space for annotation in the context of visual data-driven storytelling. Annotated charts can be readily found in scientific publications, journalistic media, information graphics, as well as in government and organizational memorandum [149]. These charts also appear in the context of live presentations [20], such as those delivered in educational or conference settings, where the speaker can elaborate further upon the charts and their annotations.

Our survey was motivated by the analysis of a corpus of such presentations from the "Quantified Self" or personal data tracking community; Choe *et al.* documented how quantified selfers visually presented their insights, which included a discussion of annotation [22]. The quantified selfers commonly used annotations to effectively communicate their personal insights. The most common annotations included text, shapes, trend lines, ranges highlighted via color or texture segmentation, and lines indicating meaningful values. To further inform our design space, we surveyed annotated charts found in journalistic media. We selected this domain for three main reasons: such charts are intended for a large audience, they are numerous and readily accessible, and most importantly, they are often intended to support or tell a story, such as by accompanying a news article.

We began by collecting a small pilot corpus of 30 annotated charts from sources including *The New York Times* and *FiveThirtyEight*. By inspecting this corpus and by considering the findings of Choe *et al.* [22], we formulated an initial design space of annotation targets and annotation forms.

Like the machine learning approach of validating whether a classification based on an initial training set generalizes appropriately, we tested against a larger representative corpus of annotated charts to validate our classification of annotation targets and forms.

We began with the collection of 705 news charts from the Massvis 2k dataset [149], which were predominantly variants of bar charts, line graphs, and scatterplots published by *The Economist* and *The Wall Street Journal (WSJ)*. We identified 257 of these charts as having some form of graphical or textual element meeting our criteria for annotation. To be clear, our aim was not to quantify the various forms of annotation or make claims about their prevalence, but to identify unique annotations. For instance, consider two bar charts published by the same news desk that use the same style guidelines; if both charts included text annotation adjacent to bars, we only retained one of these charts in our corpus. However, if one of these two charts contained an additional unique form of annotation, such as a line perpendicular to the bars indicating the average value of all bars, both charts were retained in our corpus. After discarding charts with duplicate forms of annotation, we were left with 23 *Economist* charts and 39 *WSJ* charts. Acknowledging the stylistic differences and the different forms of annotation employed by these two organizations, we then decided to gather additional annotated charts from four other news graphics desks: *The New York Times* (10), *FiveThirtyEight* (11), *The Washington Post* (11), and *Bloomberg Visual Data* (12). To ensure comparability with the *Economist* and *WSJ* charts from the Massvis dataset, we limited ourselves to variants of bar charts, line graphs, and scatterplots containing a unique form of annotation.

Altogether, we arrived at a corpus of 106 annotated charts from six sources; this corpus, labeled with our design dimensions, along with links to original sources, is available at `https://chartaccent.github.io/#section-survey`. For each chart in this corpus, we characterize its type (*e.g.*, bar chart, line graph), the annotated chart elements, annotation forms, the visual properties of annotations such as font size, color, and stroke width, along with additional comments.

4.2.2 Analysis

Informed by our survey of annotated charts, we characterize two design dimensions for annotation in the context of visual data-driven storytelling: annotation form and annotation target.

The question of why a person would annotate a chart cross-cuts these two dimensions: a particular annotation form applied to a particular target will ideally achieve a specific effect, to add interpretive value [150]. In some cases, the act of annotation is self-serving; consider a student who annotates a textbook as a means to study, or a person who annotates a calendar to serve as a personal reminder. In other cases, the act of annotation is intended to enable communication: to attract and orient the audience, to explain and facilitate interpretation [145], to draw their attention and emphasize one or more elements in a document, to separate and distinguish elements [151], and to provide context or editorial commentary with reference to these elements. Prior discussion of annotation in the visualization literature refers to both its personal and communicative purposes, and the act of annotation has often been associated with the abstract notion of "insights": manipulating them [147], externalizing them [148], as well as recording, organizing, and communicating them [73]. As the motivation for this chapter pertains to visual data-driven storytelling, we are most interested in the communicative purposes of annotation.

Dimension 1: Annotation Form

Many definitions of annotation tend to imply that an annotation is merely text commentary that has been added to a document; we find such definitions to be too narrow. According to Marshall [150], an annotation can be any kind of superstructural element added to a document, and her study of students' marginalia in textbooks suggests that

annotations can take many forms beyond mere text. In the visualization literature, annotation is a "layer of user assistance and user insight" [145], a single annotation is regarded abstractly as meta-information related to some data, and the annotation is represented with a visual object [147]. Other definitions are less abstract [73, 148], referring to graphical or textual elements added to a visualization artefact (*e.g.*, a chart, a map, a plot, a graph).

Based on our survey of annotated charts, we distinguish four forms of visual annotation: text, shapes, highlights, and images.

Text Data-driven text annotations indicate values corresponding to data-bound chart elements, such as the attribute value pair of a point in a scatterplot, or the upper and lower bounds of a range along one attribute; examples include the items in a set (Figure 4.2-c) and the average temperature for a series (Figure 4.2-h). When only a subset of data-bound chart elements is annotated, the intent is to draw the viewer's attention to them before they examine elements lacking any annotation. Other text annotations that are not data-driven can provide additional context, orientation, or editorial comment, such as in Figure 4.2-g. Text annotations also have a number of visual properties including those pertaining to font, justification, padding, wrapping, and position relative to the annotation target.

Shapes These annotations can be distinguished by their type (*e.g.*, line, arrow, curve, rectangle, ellipse, bracket, star, speech bubble), by their stroke and fill values, and by their position relative to the annotation target. Rectangles or ellipses can be placed to contain chart elements, prompting the viewer to acknowledge the importance of these elements or to compare them to elements outside of the set. Arrows, stars, and symbols can be used to direct the viewers' attention to a single element including an annotation added

to emphasize a group of elements. Like text annotations, shape annotations can also be data-driven; for instance, trend lines in a scatterplot require calculations to be performed on the underlying data, such as in Figure 4.1-bottom.

Highlights This form of annotation involves altering or embellishing the target to emphasize or diminish its importance. A highlight can be distinguished by the visual properties of the target that it alters, such as its size or its stroke and fill values. For instance, items in Figure 4.2-c have an orange fill and stroke, distinguishing them from other temperatures in the Chicago or Phoenix series.

Images Images and icons added to targets can be distinguished by their size, their opacity or saturation, their position relative to the target, including whether they are in the foreground or background. Examples of image annotations include the flags in Figure 4.1-bottom. Highly salient image annotations may be used to promote the memorability of a chart [146].

Combining annotation forms Note that any single target can be annotated with several annotations. For instance, a point in a scatterplot can be highlighted via a different fill color and stroke, it may have a text annotation displaying its value, and a dropline or arrow may originate from the text annotation to the point.

Dimension 2: Annotation Target Type

We distinguish four types of targets, where a target is the object or objects being annotated: data items, structural chart elements, coordinate spaces, and prior annotations. These target types were represented throughout our corpus of annotated charts, which consisted of variants of bar charts, line graphs, and scatterplots; as a result, our current set of targets may not account for all possible targets in other chart types.

Data item, set, and series targets These targets correspond with data: (a) a single item in the data, such as Phoenix's May temperature in Figure 4.2-a; (b) a series of items reflecting relations in the underlying data, such as Phoenix's series of average temperature values in Figure 4.2-b; or (c) a set of items, such as Chicago and Phoenix average temperatures on November and December in Figure 4.2-c. When a chart contains multiple series, a set could include items from more than one series. Additional data item targets include: (d) items that satisfy inequalities, such as temperatures below the freezing point in Figure 4.2-d; or (e) items with extreme values, such as the maximum overall temperature in Figure 4.2-e. The purpose of annotating these targets is to indicate their importance, to serve as exemplars for other nearby data items, or to orient the viewer to the chart's coordinate system.

Coordinate space targets These targets are specific to the coordinate system of the chart; we assume a Cartesian coordinate system with one or two quantitative scales, which is reflected in our survey of annotated charts. Coordinate space targets include: (f) those that refer to a value such as the freezing point of 32° in Figure 4.2-f; (g) a span along an attribute, such as between March and May in Figure 4.2-g; or (h) a minimum, mean, median, or maximum value for a particular series, such as the average Chicago temperature in Figure 4.2-h.

Coordinate space targets also include those that refer to value pairs or spans along two attributes: (i) a point, such as 10° in August (Figure 4.2-i); (j) a partial span at a specific attribute value, such as 10° in a range between October and December (Figure 4.2-j); or (k) a span along two dimensions, such as the region in Figure 4.2-k. Shape annotations for coordinate space primarily serve to provide context and orientation to the viewer, to emphasize important positions or ranges, including those that contain no data items.

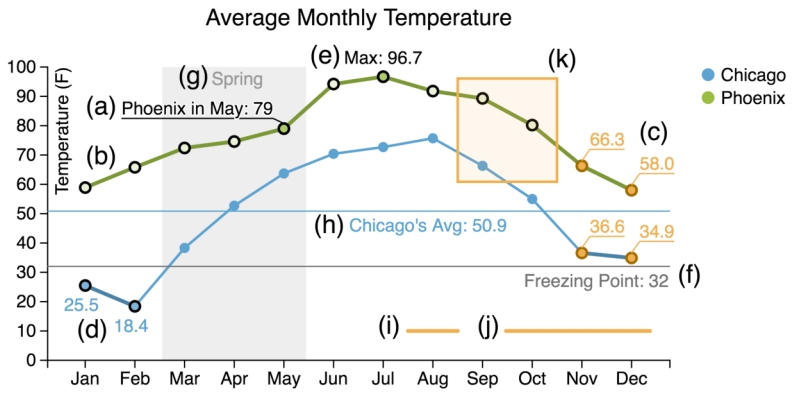


Figure 4.2: An example line chart of average monthly temperatures indicating the types of data item, set, & series targets (a–e) and coordinate space targets (f–k); the parenthetical labels (a–k) are annotations used to reference prior annotations.

Chart element targets These targets include the chart title, the axes, the axes labels and tick marks, the legend, the plot area of the chart, and finally the chart itself. It is important to distinguish the final two target types in this list: the plot area refers only to the area encompassed by the coordinate system indicated by the axes, whereas the entire chart encompasses all of the preceding elements in this list. For instance, a text annotation within the plot area may provide additional context about the data or help orient the viewer by explaining the choice of scale or range. In contrast, an annotation on the entire chart typically appears on the periphery of the chart, encompassing captions, credits, and footnotes, providing additional context information or text that would be too verbose to include within the plot area.

Prior annotations Annotations are by definition additive, and thus a previous annotation can be the target of additional annotations. For example, the parenthetical labels (a–k) in Figure 4.2 are annotations used to reference prior annotations.

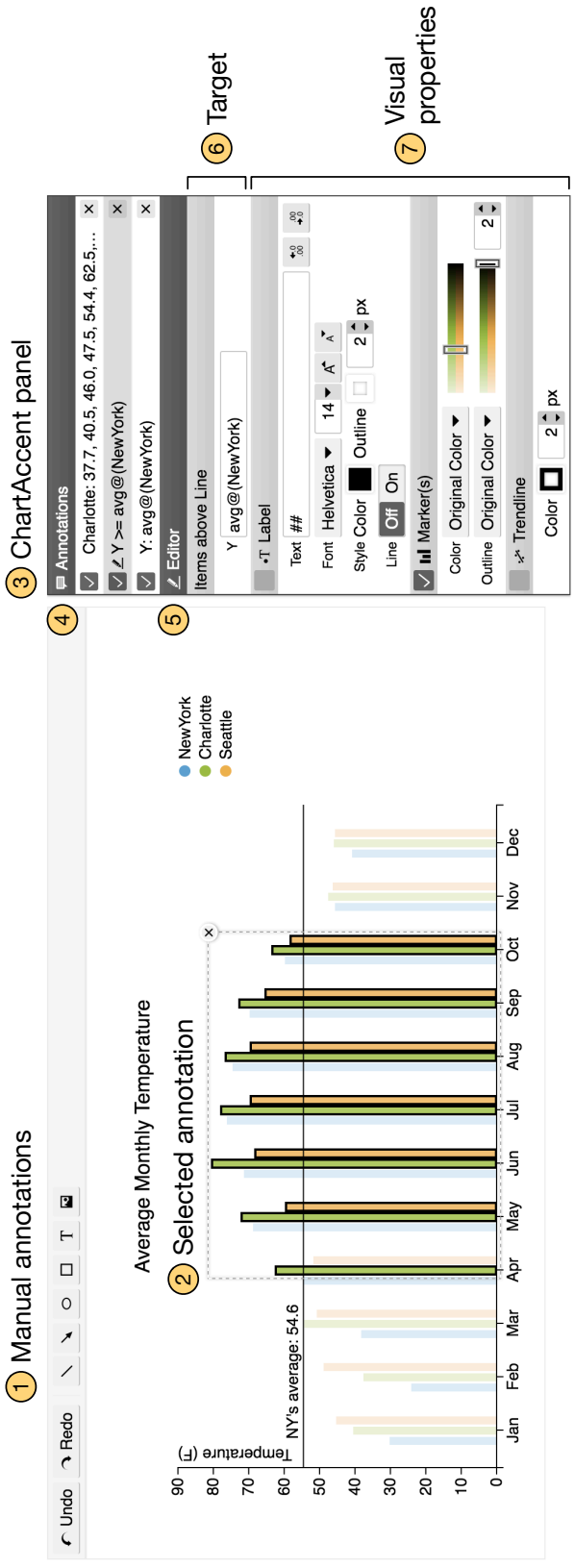


Figure 4.3: ChartAccent's user interface; (1) manual annotations; (2) the selected annotation represented by a gray dotted rectangle; (3) the ChartAccent control panel; (4) the list of annotations currently applied to the chart; (5) the annotation editor, which allows one to modify (6) the annotation target; and (7) the visual properties of the selected annotation.

4.3 ChartAccent

We used our design space to design and implement ChartAccent, a tool that provides a palette of interactions for the data-driven annotation of a chart. This section contains our design rationale and ChartAccent's treatment of annotation forms and targets.

4.3.1 Interacting with ChartAccent

We decided to separate interactions for annotation from those for chart creation. This has two benefits: (1) as a research prototype, it allows us to evaluate annotation interactions independently; (2) the interactions we designed for ChartAccent are transferable and can be ported to other systems used to create charts. There are two classes of annotation interactions: those that specify the annotation target(s), and those that specify or modify the annotation form.

Annotation Target Selection

Following a typical user interface selection mechanism, mouseover highlights the target to be annotated, and mouse click selects and annotates the target; a mouse drag, lasso, or click while holding a modifier key results in a selection for either a coordinate space target or data item target.

ChartAccent includes an implementation of bubble cursor selection [152]: when approaching a data item target (*e.g.*, a series in a line chart), the cursor passes an initial distance threshold and the entire series becomes a selection target; after a second threshold closer to the line, either a data item (an inflection point in the line) or a line segment and the two data items that it joins becomes the selection target, depending on which is closer to the cursor. Meanwhile, a coordinate space target can be selected by clicking a position along an axis or by dragging over a span along an axis.

Data-driven selection ChartAccent adopts aspects of Heer *et al.*'s selection design [74], in which a person can click on a legend item to trigger annotations for a series of data items that share the selected categorical value.

Once the default annotation for a selected target appears, the position or span can be adjusted using ChartAccent's target editor (Figure 4.3-6); this novel counterpart to direct target selection allows a person to compose basic formulas that can reference the attributes of the dataset. The editor features auto-complete support and includes a set of basic statistical functions including minimum, maximum, mean, and median. For instance, the horizontal line annotation at $y = 54.6$ in Figure 4.3 representing New York's average temperature was positioned using such a formula.

Upon selection of a coordinate space target, the target editor provides a novel mechanism to select additional targets relative to these positions or spans: the ability to select data items above or below a position or within or outside a span without having to manually click or drag over them; the Charlotte and Seattle monthly temperature values higher than the New York average in Figure 4.3 were selected in this manner. This selection is tightly coupled with the originally selected range target; if the line or rectangle annotation corresponding to the original coordinate space target is adjusted, ChartAccent automatically updates any associated selections of data items.

Annotation Form Specification & Modification

As we encountered in our survey of annotated charts, annotation can take many forms and the visual properties of each form can vary tremendously. With ChartAccent, we established a default annotation form for each data item and coordinate space target type following its selection. The default annotation for an individually-selected data item is a black stroke border and a text annotation indicating the item's value, which appears adjacent to the item. The default annotation forms for a coordinate space targets are

lines or rectangles drawn perpendicular to the axis from the selection position or span, respectively, and a text annotation indicating the values (Figure 4.2-g–h).

Annotation forms for set and series targets When a selection involves more than one data item target, a trend line can be added to the selected set or series, as shown in Figure 4.1-bottom and Figure 4.6-4. Additionally, when a set or series are selected in a scatterplot, a Bubble Set [153] annotation can be added to the contour surrounding these targets, as shown in Figure 4.1-bottom and Figure 4.6-5.

Manual annotation forms Chart element targets can be manually annotated by selecting a form (Figure 4.3-1) and positioning the resulting annotation anywhere on the chart via dragging; lines, arrows, ellipses, rectangles, text, and images can be used to annotate chart element targets. Coordinate space targets that do not intersect an axis (such as Figure 4.2-i,j,k) can also be annotated in this manner.

Annotation form modification Once added to a chart, the visual properties of the annotation can be interactively modified. The position and size of existing annotations can be adjusted via dragging. Meanwhile, ChartAccent's control panel for modifying the visual properties of annotations mimics popular graphical tools (Figure 4.3-7). Annotations for sets and series of data items share the same visual properties, and thus each item does not need to be modified individually. Annotation properties include font properties such as type, size, and color, the visibility of a dropline connecting an annotation to its target, as well as stroke and fill properties.

4.3.2 Usage Scenarios

We illustrate the process of annotating charts with ChartAccent through two usage scenarios, which are also showcased in the supplementary video.

Scenario 1: Monthly Temperature Line Chart

In our first scenario, we consider average monthly temperatures for several American cities from July 2014 to June 2015 (data from FiveThirtyEight [154]). Let us imagine a situation in which we want to convince others that the weather in New York is generally not as favourable as the weather in Charlotte or Seattle. To support this argument, we will highlight the months when the temperature in Charlotte and Seattle is higher than New York's average temperature with a grouped bar chart.

We begin by diminishing the highly salient bars to provide a greater contrast for the bars that we want to highlight. To do so, we select all the bars by dragging over all of them. We then diminish the salience of the bars by adjusting their brightness in the visual properties editor (Figure 4.3-7).

Next, we indicate the average temperature in New York. To do so, we select a range target by clicking on a point along the Y axis, which results in a horizontal line annotation. Then, in the target editor (Figure 4.3-6), we replace the target value with `avg@(NewYork)`. We then prepend "NY's Average: " to the default label and drag it to the left side of the chart, as in Figure 4.3.

To highlight the months when Seattle and Charlotte's temperatures are above New York's average, we open the "Select Items Using this Line" dropdown menu, select "Above," "Charlotte," and "Seattle," leaving "New York" unselected. Once selected, black strokes and text annotations appear for each bar meeting this criterion; once again, we toggle off the visibility of the text annotations, but we keep the black strokes for the selected bars.

Finally, we export the resulting annotated chart (Figure 4.1-top) as a PNG or SVG image.

Scenario 2: Fertility Rate vs. Life Expectancy Scatterplot

In our second scenario, we consider fertility rate and life expectancy in several countries as of the year 2000 (data from Gapminder [155]). In particular, we want to (i) convey the overall trend that higher fertility rate is correlated with lower life expectancy, (ii) highlight North and South American countries, and (iii) highlight the United States, China, and India.

We begin by creating a scatterplot with these two attributes where countries are color-coded by world region and sized by population. Much in the same way that we diminished the salient bars in our first scenario, we diminish the fill value of the points. We then add a trend line by toggling trend line visibility (Figure 4.3-7). We manually add a text annotation to explain the trend line, as well as an arrow shape annotation to establish a visual connection between this text and the trend line.

To highlight countries in North and South America, we click on the "America" legend item to indirectly select all of the corresponding data items. We toggle off the visibility of the text annotations and we turn on the bubble set annotation for this selection.

To highlight United States, China, and India, we repeat the following steps for each country: selecting each country and moving the default text annotation (*i.e.*, country name) to a nearby location where it does not occlude other points, then manually adding and positioning each country's flag as an image annotation. Since the text annotation for the United States is far from its corresponding point, we trigger the visibility of a dropline from it to the point.

Finally, after adding a text annotation indicating the year "2000," we can export the resulting chart (Figure 4.1-bottom).

4.3.3 Implementation Details

We initially implemented the ChartAccent.js library to support annotation on SVG-based charts (such as those generated using D3.js [7]). Figure 4.4 shows the relationship between ChartAccent.js, a chart creator, a chart, and an end-user. To make a chart "annotatable," the chart creator loads the data and renders the chart. The chart creator must also create a "ChartAccent object," register chart elements such as axes, marks, and legends as well as their metadata, and map marks with their corresponding data items as well as their default annotation form. ChartAccent.js also has functions for creating and managing annotation layers and their visual properties, as well as interaction handlers for the various forms of annotations.

ChartAccent.js supports charts comprised of the following elements: Cartesian coordinate systems of linear or ordinal scales, circle or rectangle marks, polylines connecting marks within series, and bullet-style series legends. ChartAccent.js thus can be used to annotate many variants of bar charts, line graphs, and scatterplots. For other types of charts, ChartAccent.js currently provides limited support for item-based annotations (such as in the annotated node-link graph and treemap featured in Figure 4.5).

Finally, ChartAccent itself is a standalone tool that encapsulates a basic chart creator interface and the ChartAccent.js library, which includes an interactive panel for modifying annotations and their visual properties (Figure 4.3-3).

4.4 Evaluation: Reproduction Study

To evaluate the experience of interacting with ChartAccent to annotate different types of charts encompassing a variety of annotation targets and forms, we asked participants to reproduce a set of previously annotated charts, beginning with unadorned (*i.e.*, unannotated) versions of the same charts.

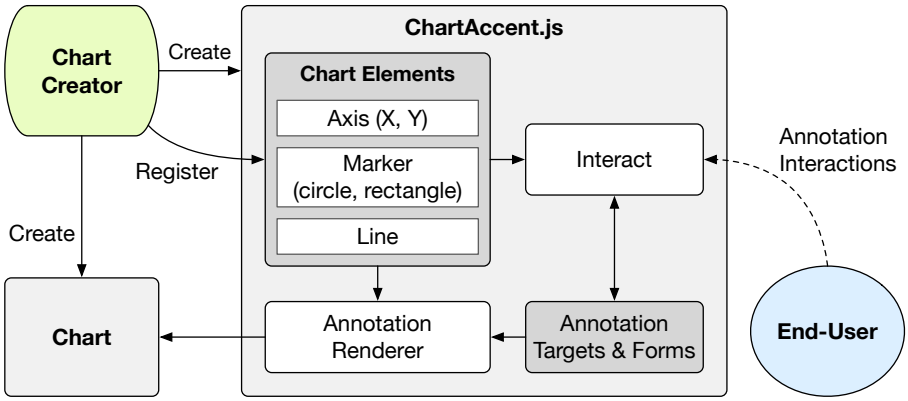


Figure 4.4: The ChartAccent architecture; the chart creator program that loads the data and renders the chart creates a ChartAccent object and registers chart elements; ChartAccent.js includes facilities for managing annotation layers and rendering an internal annotation representation to the chart in response to an end-user's interaction.

The major question that motivated this evaluation was whether the set of interactions to support annotation in ChartAccent were learnable, usable, and efficient. We were particularly curious about data-driven annotation via direct and indirect selection, as the combinations of these features set ChartAccent apart from previous approaches for annotating charts.

4.4.1 Participants and Setup

We recruited 11 (3 women, 8 men) participants from the Greater Seattle area. All of our participants had created basic charts (*e.g.*, bar charts, line graphs, pie charts, etc.) using commercial tools within the past three months; they also stated that they had previously used charts for communication purposes, such as in a live presentation. These participants had normal or corrected-to-normal vision, however one participant was color-deficient but not color-blind. Various occupations were represented among our participants, including a receptionist, a real estate broker, a business analyst, a software engineer, and a user experience researcher. The average age of our participants was 35, ranging from 23 to 50 years of age. Participants were compensated with a software

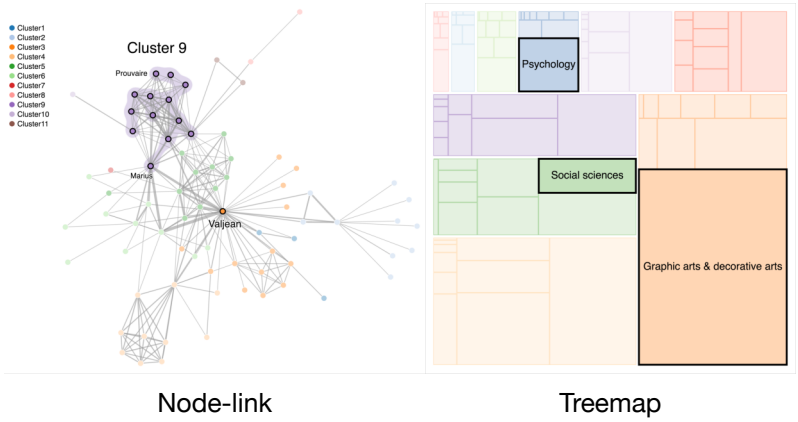


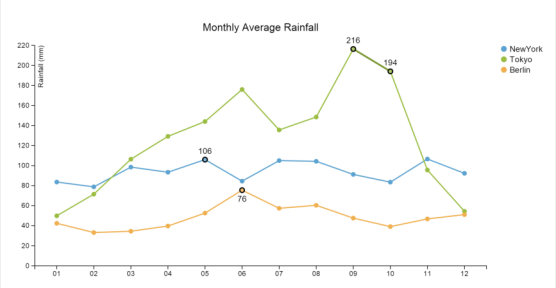
Figure 4.5: Data item annotations in a node-link graph and a treemap. Left: character co-occurrence graph in *Les Misérables*, with a bubbleset-highlighted set ("Cluster 9") and 3 other nodes selected, with other nodes diminished in salience (data from [156]). Right: checkout count by Dewey category from Seattle Public Library, highlighting three items and diminishing the salience of others (data used with kind permission from George Legrady).

gratuity.

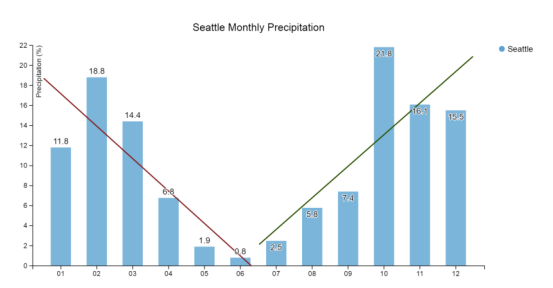
We used a 3.6 GHz Windows 8 desktop machine with 32 GB RAM, using two side-by-side 24-inch Dell LCD displays running at 1920 × 1080 resolution; the left monitor was in a portrait orientation. For all tasks, we logged start, reset, and end times, and we saved the result annotated chart as an image. We also captured screen recordings along with concurrent video and audio recordings of the participants as they interacted with ChartAccent.

4.4.2 Tasks and Datasets

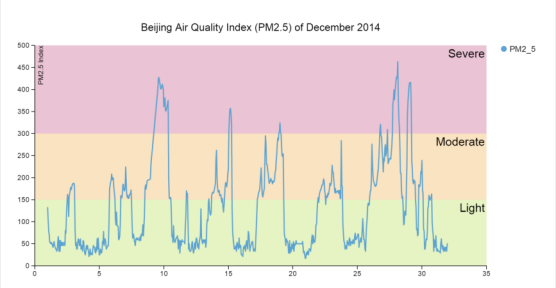
We prepared seven annotated charts using three chart types: Tasks 1 and 3 featured a line graph, Tasks 2 and 7 featured a bar chart, and Tasks 4–6 featured a scatterplot, as shown in Figure 4.6 for Task 1–6 and Task 7 (Figure 4.1-top and Figure 4.3, Scenario 1): Select all series and diminish their salience; add line annotation to a coordinate space target corresponding to average New York temperature; select and annotate values above this



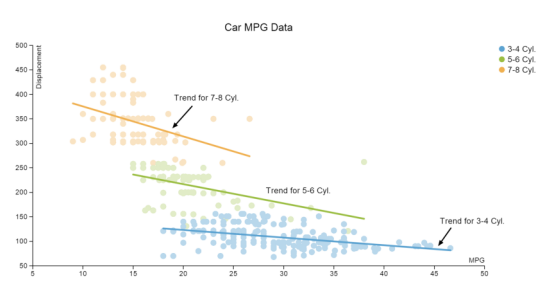
Task 1: Select and annotate four individual data items; drag text annotations to correct positions.



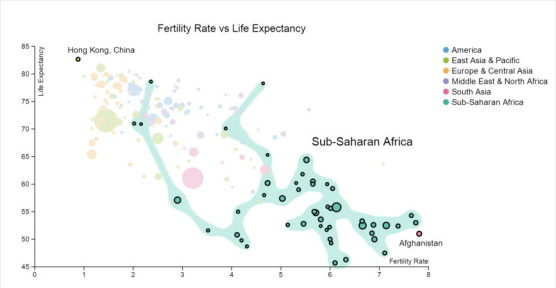
Task 2: Select and annotate two sets of data items; remove stroke highlights; add trend lines for both sets; modify trend line colors.



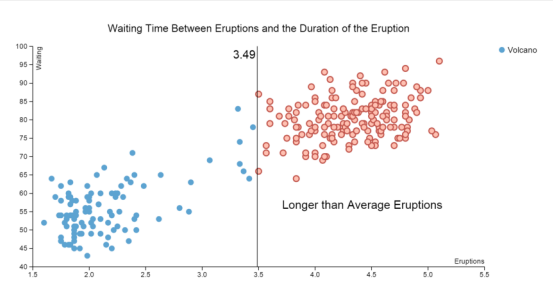
Task 3: Annotate three coordinate space targets; modify fill color of range annotations; edit text labels for these coordinate space annotations.



Task 4: Select each series; diminish their salience; add trend line for each series; add and position text and arrow annotations.



Task 5: Select all items and diminish their salience; annotate a series and four individual data items; modify and drag text annotations to correct positions.



Task 6: Add a coordinate space annotation at the average x value; select and highlight all data items with a greater x value; add and position a text annotation.

Figure 4.6: Reference annotated charts used in the study for Tasks 1–6.

value; select and annotate a single data item; edit and position text annotation. Step-by-step animations of these tasks are available on the supplemental website. These tasks and the corresponding annotated charts were selected to encompass ChartAccent's capabilities and featured a variety of annotation forms and targets. Each subsequent task/chart increased in terms of complexity: from individual data items to set and series targets, the annotation of coordinate space targets, the combination of data-driven and manually-added annotation, and an increasing number of visual property modifications. Figure 4.6 summarizes the steps required to complete each task, and a step-by-step animated GIF for each task is available at `https://chartaccent.github.io/#section-examples`. We also prepared an additional twelve annotated charts for practice tasks.

The datasets to create these charts included those pertaining to: monthly average rainfall in various cities [157] (Task 1); monthly precipitation in Seattle [157] (Task 2); the Beijing air quality index [158] (Task 3); mileage statistics for various cars [159] (Task 4); fertility rate and life expectancy in various countries [155] (Task 5); Old Faithful Geyser eruptions [160] (Task 6); and monthly temperatures for major American cities [154] (Task 7).

4.4.3 Procedure

We began with a brief explanation of the study goals and overall procedure. We then asked the participants to complete a pre-study background questionnaire, and led them through a tutorial on the core concepts and features of ChartAccent, with an emphasis on coordinate space targets as well as the combination of data-driven selection via direct manipulation and indirect selection via the target editor; participants were encouraged to interact with the tool during this tutorial. On average, the tutorial lasted 36 minutes.

Following the tutorial, the participants performed twelve practice tasks to familiarize

themselves both with the task procedure and with ChartAccent. For each task, we showed a static annotated chart as a reference on the left monitor, and asked the participants to reproduce the same chart on the right monitor with ChartAccent, which was embedded within a study management application. We also showed the original chart without any annotations as an image below the annotated chart as a reference.

Before starting each task, we asked the participants to verbalize the required annotations based on what they saw in the annotated reference chart; we did so to ensure that the participants did not miss any annotations from inattention, and to provide them with an opportunity to seek clarification regarding the properties of each annotation, as properties such as line thickness and fill color were not always clearly distinguishable, especially for small targets. After we confirmed that the participants understood all of the required annotations, we asked them to press a "Start" button to load the editable version of the chart and begin the task. We also asked them to press a "Submit" button after completing the task, which would save the resulting chart as an image and advance to the next task. We allowed the participants to press a "Reset" button, which would remove all annotations from the chart and restart the task (but not the timer). We encouraged the participants to think aloud, especially when any aspect of the task or ChartAccent was confusing or unclear. On average, the time to complete the twelve practice tasks was about 22 minutes.

After completing the 12 practice tasks, the participants repeated the same procedure with Tasks 1–7. We provided hints to the participants either when their progress stalled or when they tried to submit the results with incorrect or missing annotations; for the latter, we pointed out the errors and asked the participants to fix them. We noted the cause of the stall or error in both cases. We encouraged the participants to complete the task as quickly as possible, and explained that they did not have to match the exact x,y position or color of annotations in the reference chart, which would have been especially

tedious for text annotations. On average, the time to complete all seven tasks was about 15 minutes.

At the end of the study session, participants filled out a questionnaire regarding their experience with ChartAccent. On average, the study session lasted about 1.5 hours.

4.4.4 Results

Nine out of 11 participants successfully reproduced the annotated charts for all seven timed tasks; the two remaining participants arrived late and ran out of time, and thus could only complete the first five tasks. Because we asked the participants to fix any errors without imposing a time limit, all of the resulting annotated charts they created were correct copies of the reference charts.

Hints and task completion time As mentioned in our description of the procedure, we provided two types of hints to the participants. First, we explained what went wrong (*e.g.*, the z-order of annotations prevented the addition of another annotation) or reminded the participants of ChartAccent's capabilities (*e.g.*, indirect selection and annotation of data items relative to a coordinate space target via the target editor). Second, we pointed out the difference between the reference chart and the participants' outcome (*e.g.*, missing text annotations or the addition of unnecessary annotations), which was usually caused by oversight. Table 4.1 shows the number of hints we provided for each task. Overall, participants successfully completed the tasks with very few hints, and they needed more hints for minor mistakes. The most hints were given during Task 5, which was similar to Scenario 2 and required a combination of data-driven annotations, manual text annotations, and form modifications.

With regards to task completion time, the overall average task time across all tasks and participants was 102.7 seconds (see Figure 4.7), indicating that charts of varying

	# Hints: features & concepts		# Hints: missing, incorrect annotations	
Task	**Total**	**Average**	**Total**	**Average**
T1	0	.00	4	.36
T2	6	.55	10	.91
T3	3	.27	1	.09
T4	5	.45	7	.64
T5	13	1.18	16	1.45
T6	3	.33	7	.78
T7	8	.89	3	.33
Total	38		48	

Table 4.1: Total and average number of hints per task. (P1 and P5 completed only the first five tasks.)

complexity requiring several steps can be completed quickly. The two longest task completion times (P4 and P7 in Task 5; see Figure 4.7) were due primarily to a z-order issue: *"The steps to hiding/obscuring and lines, if done out of sequence, it can be a little frustrating"* (P4). Some participants were unaware of the consequences of deleting an annotation, as sometimes they deleted an annotation with the intention of merely hiding it. This issue caused some delay in completion time when the form of the deleted annotation had been modified.

Participant feedback Participants rated ChartAccent on four satisfaction criteria. All ratings were on a 1–7 Likert scale from "Strongly Disagree" to "Strongly Agree." Overall, ChartAccent was rated fairly highly; participants deemed it easy to learn (Avg = 5.8) and use (6.2), they found the annotation creation process to be enjoyable (6.7), and they indicated a desire to use ChartAccent to annotate charts in the future (6.9).

In a post-study interview, participants elaborated on the usefulness of ChartAccent: *"Very cool idea, I can see it being very useful for presenting data. [I] definitely can think of times where I wish I had this kind of tool to make my presentations easier to make, and better overall."* (P8); *"I liked [that] it was semantic based. ... Overall I liked it a lot actually.*

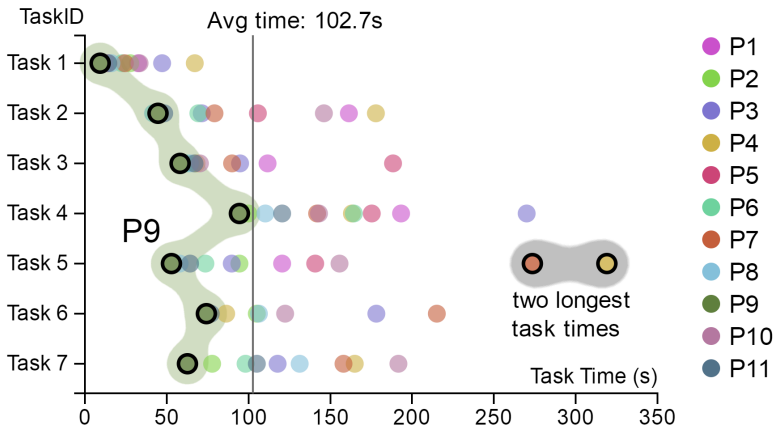


Figure 4.7: Task completion time by participant and by task; P9 (a software engineer and prolific maker of charts) had the shortest completion time for most tasks. (This figure was generated with ChartAccent.)

Because it's dynamic, for the most part, it means that I will be able to modify values and the annotation might be able to follow the similar rules right out of the box. That's cool." (P11 referring to ChartAccent's data-driven annotation and the target editor in particular); *"No more photo editing! This will save me time."* (P4). Participants also mentioned how ChartAccent was easy to learn: *"[there is] good indication about what will be selected, straightforward interface (very similar to existing products) making it easy to learn"* (P2); *"It had [a] familiar interaction language that I'm used to, so learning was relatively easy and using it was intuitive."* (P9, a software engineer who is familiar with Adobe Photoshop). On the other hand, several participants noted learnability issues: *"There was a lot to remember."* (P5, who was unfamiliar with marquee/lasso and keyboard shortcut conventions).

4.5 Discussion and Future Work

We now discuss the lessons learned during the design and evaluation of ChartAccent, along with directions for future work.

4.5.1 Reflection on the Survey of Annotated Charts

Our work was initially inspired by the analysis of annotated charts used personal data presentations by "quantified selfers" [22], which investigated *what* people annotate in terms of personal insights and *how* they annotate those insights in terms of annotation forms. To validate and build upon this analysis, we surveyed annotated charts found in journalistic media; these charts are targeted for a broader audience and created by professionals for asynchronous reading. We observed a similar annotation targets and annotation forms in charts emanating from these two domains.

While our analysis of annotated charts from these domains may not fully encompass all possible existing forms of annotation, they helped us distill a design space, which in turn informed the design of ChartAccent. In the future, it would be helpful to compare against annotated charts from other domains, such as those used in public policy, corporate governance, and education.

4.5.2 Improving and Extending ChartAccent

In the months following the reproduction study, we addressed usability issues that we identified during the study, including the ability to adjust the z-order of annotations by reordering the annotation list (Figure 4.3-4), and by adding basic undo and redo functionality.

On a larger scale, the question of how to enable people to effectively construct charts and other visualization artefacts is an ongoing research problem [49]. Instead of building a comprehensive chart authoring system, we decided to focus on developing and testing a set of interactions to support chart annotation. In addition, we ensured that our ChartAccent.js library would be compatible with existing charts generated with D3.js [7] and would not be limited to charts created with the current standalone ChartAccent tool. As

indicated in Figure 4.4, we envision cases in which chart creators use function calls to register chart elements created with D3.js. One way to achieve this goal could involve incorporating D3 Deconstructor [161] for automatically determining the underlying data and the visual mappings of a chart, thus enabling ChartAccent to import existing charts without having to specify the chart elements.

Currently, ChartAccent supports variants of bar charts, line graphs, and scatterplots. Although ChartAccent can be used to add annotations to data item targets in other chart types, such as node-link graphs or treemaps (see Figure 4.5), current support for charts that do not have a Cartesian coordinate system is limited. Future work remains to extend ChartAccent to other coordinate systems and chart types.

4.5.3 Desirable Annotation Form Defaults and Templates

In our reproduction study, we learned that the default annotation forms applied by ChartAccent were often not ideal for the tasks that we had selected, which were intended to assess a broad range of features and required several steps to complete. We note that, in general, desirable default annotation forms depend upon context and personal preference. For instance, each newsroom has a style guide for charts, and the chart author may have annotated similar charts of the same type in the past in accordance with this style guide; in such cases, selecting the form of annotations and modifying their visual properties might become very tedious. We plan to allow people to configure default annotation forms instead of redesigning ChartAccent to best support the particular tasks that we examined in our reproduction study.

Furthermore, as P11 acknowledged, annotation interactions can be reused, as in the case of Photoshop's "Action" feature: people can record interactions and replay the steps to complete the tasks automatically. These annotations could also be saved as a template

and applied to an updated dataset having the same schema. For example, in Scenario 1, we could highlight the months with temperatures higher than New York's average for every year by applying the same annotation template. For additional flexibility, data-driven annotations can be applied to a different dataset with a similar schema by devising an annotation specification language. As the selection and the visual properties of annotations can become complex with many annotations, it would be helpful (if not necessary) to have an editor for annotation specification templates. We view the development of such a language and corresponding template editor as important future work.

4.5.4 Study Limitations and Web Deployment

As a preliminary evaluation, the main goal of the reproduction study was to assess the usability and learnability of ChartAccent and to assess if people could reproduce examples efficiently without excessive prompting or aid from the researchers.

We therefore specified the set of tasks used in our usability study such that they would encompass the broad range of features and interactions introduced in ChartAccent, such as those related to data-driven selection via direct manipulation and indirect selection using the target editor. However, since the charts and the data they depict were not personally meaningful to the study participants, we cannot assume that they were highly engaged and motivated to annotate these charts. As a result, we have yet to truly assess the expressiveness of ChartAccent; as a next step, we plan to study the annotation of charts containing data provided by participants.

Furthermore, though we emphasized to our participants that we were evaluating ChartAccent and not their performance, any study protocol where a researcher is observing over their shoulder and pointing out mistakes may have imposed a high level of stress. We suspect that this pressure may have caused some of the errors or obsessive detail-

checking, inflating our task completion times. With respect to their subjective feedback, we acknowledge the possibility of a social desirability bias, particularly because we did not perform a comparative evaluation between alternative approaches.

After refining ChartAccent, we made ChartAccent available on the web: `https://chartaccent.github.io`. People can use ChartAccent with their own data, add personally meaningful annotations, and share the resulting annotated charts. ChartAccent allows people to create an annotated chart via three steps: 1) import the data, 2) select a chart type and configure chart axes, and 3) annotate the chart. In addition, it allows them to export the annotated chart either as a static image (in PNG or SVG) or as an animated GIF, which illustrates the step-by-step annotations. To assess post-deployment adoption, we plan to monitor usage as well as collect and analyze the annotated charts people created with ChartAccent as a way to understand *how* people use ChartAccent, *what* they annotate, and how we can further improve the annotation experience.

4.6 Conclusion

In this chapter, we reflected upon annotation in the context of visual data-driven storytelling. To provide empirical knowledge on the spectrum of annotations used in data-driven storytelling, we characterized a design space of annotation informed by a survey of 106 annotated charts. Using this design space, we designed and implemented ChartAccent, a tool that provides support for manual and data-driven annotation. ChartAccent provides a novel way of defining and using data-driven annotations, and we envision that our annotation interactions can be integrated into other charting environments. To evaluate the experience of creating annotations with ChartAccent, we found that most participants could easily reproduce a series of previously annotated charts using ChartAccent after a short tutorial and practice session. We also found that Chart-

Accent's design and interaction provides an enjoyable experience, and all of our study participants expressed a strong desire to use ChartAccent to annotate charts in the future. The ongoing deployment of ChartAccent will help us better understand how people annotate charts as a means of telling stories with data.

Chapter 5

Charticulator: Interactive Construction of Bespoke Chart Layouts

5.1 Introduction

The ability to create a highly customized visual representation of data, one tailored to the specificities of the insights to be conveyed, increases the likelihood that these insights will be noticed, understood, and remembered by its audience [149]. This expressiveness also gives the author of this visual representation a competitive advantage in a landscape awash in conventional charts and graphs.

However, most interactive charting tools ask chart authors to choose from a collection of standard chart types or templates, such as bar, line, or pie charts, and they provide limited customization options beyond the choice of chart type. Besides interactive charting tools, people also create charts via manual illustration or programming. Illustration

The contents of this chapter have been previously published in *IEEE Transactions on Visualization and Computer Graphics*. © 2018 IEEE. Reprinted, with permission from Donghao Ren, Bongshin Lee, and Matthew Brehmer, *Charticulator: Interactive construction of bespoke chart layouts*, *IEEE Transactions on Visualization and Computer Graphics* [12], Aug. 2018.

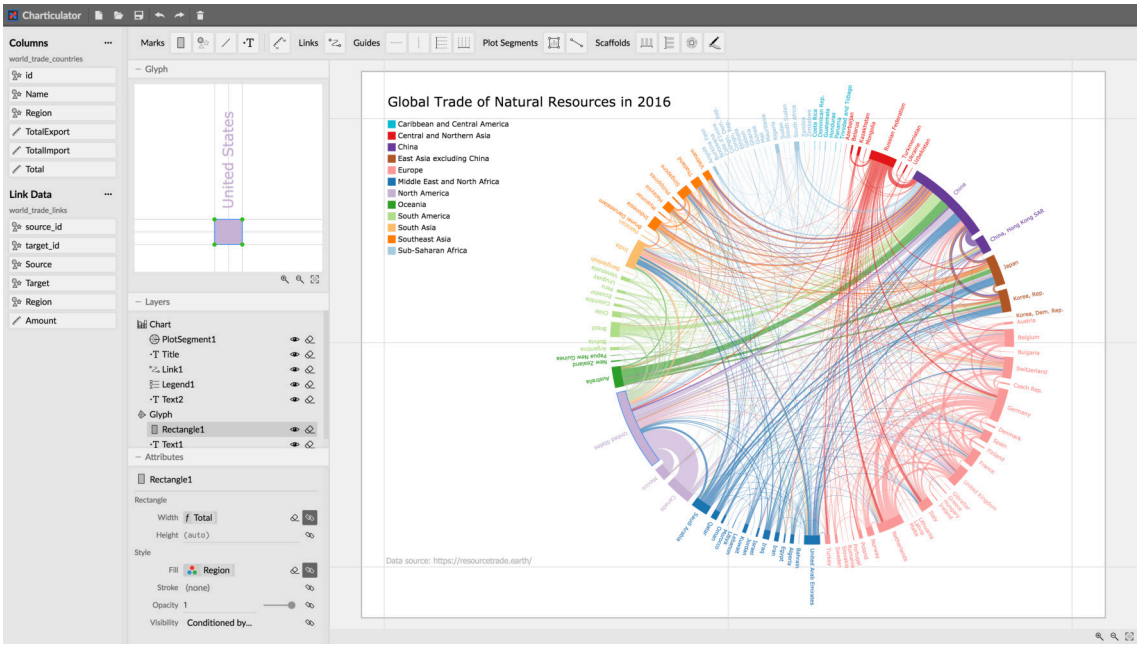


Figure 5.1: Charticulator enables a chart designer to interactively create a custom chart layout. It transforms a chart specification into mathematical layout constraints and automatically computes a set of layout attributes that satisfy the constraints using a constraint-solving algorithm. (Data from `https://resourcetrade.earth`)

tools such as Adobe Illustrator are insufficient for authoring bespoke charts because they cannot bind multiple attributes of data to graphical elements. Meanwhile, programming a bespoke chart using a library such as D3.js [7] or a declarative language such as Vega [162] provides considerable control over the encoding of data to graphical marks and their layout. This approach, however, is accessible only to a small group of people who have advanced programming knowledge.

In recent years, researchers have revisited the prospect of creating bespoke charts via interactive authoring, with tools such as Lyra [8], iVisDesigner [10], iVoLVER [163], and Data Illustrator [67]. Lyra and iVisDesigner use data-flow models to represent charts. Complex layouts in these models are modeled as data transforms that compute layout parameters. This makes layout construction a matter of choosing the optimal series of layout transforms and applying them to the data. These layout transforms are represented

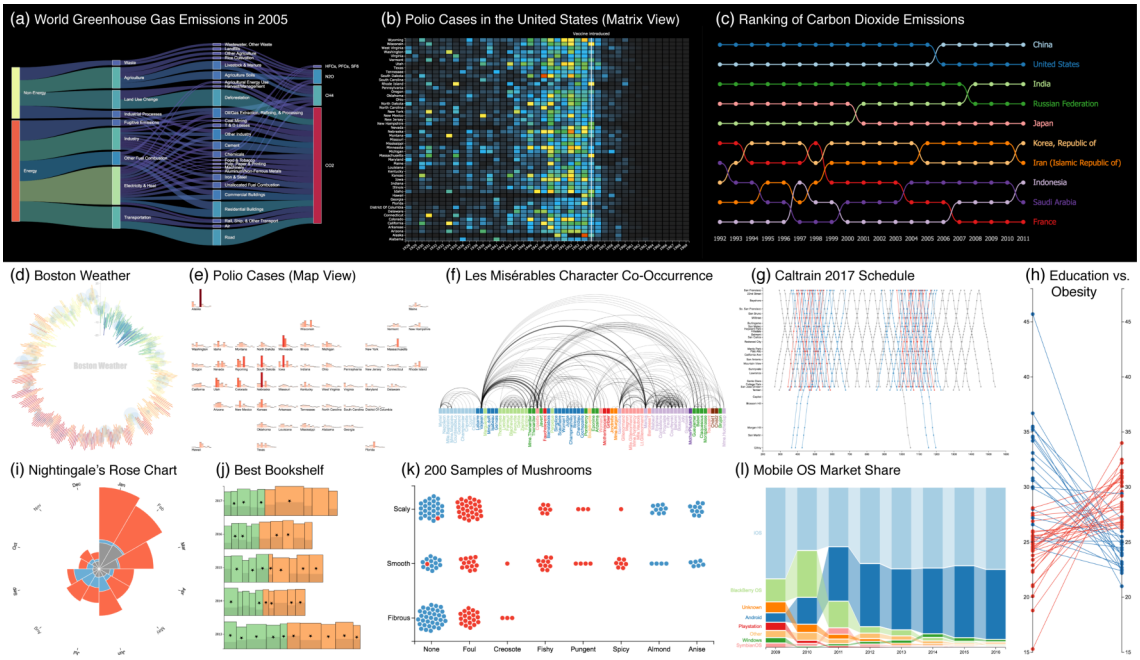


Figure 5.2: A gallery of 12 visualization examples showing the expressiveness of Charticulator. Here we hide legends and chart titles for compact presentation. More examples along with high resolution images, detailed descriptions, and videos illustrating the creation processes can be found in the website (`https://charticulator.com`). Data Source: (a) [164]; (b) [165]; (c) [166]; (d) [167]; (e) [165]; (f) [168]; (g) [169]; (h) [170]; (i) [171]; (j) [172]; (k) [159]; (l) [173]. Visual Design Source: (b) [174]; (d) [175]; (g) [176].

as configuration panels in the user interface. It is tedious to switch between layouts and exceedingly difficult to compose layouts from scratch. iVoLVER follows a bottom-up constructive approach where charts are constructed by creating graphical elements and data-calculation elements individually. However, for a chart with more than a dozen elements, this becomes very tedious. Furthermore, iVoLVER currently lacks the ability to lay out elements except for simple alignment and positioning according to (x, y) coordinates. Data Illustrator introduces the concept of repetition and partition, and supports basic layout configurations in the repetition or partition groups, combining bottom-up and top-down approaches. However, Data Illustrator can neither produce non-Cartesian charts nor charts featuring visual links including node-link graphs.

In this chapter, we present Charticulator (Figure 5.1), an interactive chart authoring tool that addresses the limitations of existing approaches by prioritizing the articulation of chart layouts as well as the visual linking between glyphs. Charticulator allows authors to specify chart layouts interactively in lieu of programmatically specifying data transformations. It then converts author-specified layouts into mathematical constraints, and employs a constraint solver to realize the chart. Furthermore, Charticulator supports exporting chart designs into reusable templates, which can be imported into other systems to visualize other data. As Charticulator leverages a constraint-based layout specification, exported templates correctly respond to canvas size changes and different distributions of data. The key research contributions of this work are as follows:

- The design framework of Charticulator, which can be applied to generate a variety of reusable chart layouts.
- The implementation of Charticulator, which realizes the design framework by transforming the chart specification into layout constraints and incorporating a constraint-based layout algorithm, with a user interface that enables interactive chart layout specification.
- Results from three forms of evaluation: a gallery of charts to illustrate Charticulator's expressivity (Figure 5.2), a chart reproduction study, and a click-count comparison against three existing tools.

5.2 Charticulator

In this section, we first present our design principles for Charticulator and its conceptual framework. We then describe its user interface and interactions along with two usage scenarios, and explain our constraint-based layout approach.

5.2.1 Design Principles

Charticulator is designed for people who lack programming skills, which may include designers, journalists, and analysts. To support a wide variety of chart layout designs, we identified the following three guiding design principles.

Promote layout as a deliberate design choice In existing visualization authoring tools, layouts are specified either via a set of predefined templates or by binding data to the attributes that affect layouts (*e.g.*, x and y position, width and height of a rectangle mark). To facilitate the creation and manipulation of diverse layouts without programming, Charticulator treats layout as a first-class citizen and exposes layout in the user interface.

Compose a layout using a set of partial specifications To enable a flexible way to specify layouts, we break layout specifications into composable parts. For example, the chart in Figure 5.1 incorporates a polar coordinate layout where wedge shapes are laid out along the circle. The spans of the wedges are bound to a data value. The text labels are positioned at the outer-middle point of the wedges. To accomplish this, we allow for a small set of partial layout specifications to be combined to produce a variety of complex layouts.

In addition, to support expressive glyph design and custom layout of these glyphs, we need to be able to express layout relationships within and between glyphs. To achieve this, we divide layout specification into two levels: chart-level and glyph-level. Charticulator presents partial specifications as objects that can be individually manipulated, and changing one part does not require chart authors to re-specify other parts. To reduce clutter, Charticulator provides two separate editing canvases: one for glyph-level editing and second for chart-level editing.

Balance direct manipulation and configuration panels Most existing vector graphics authoring tools incorporate direct manipulation of objects on a canvas, and those familiar with these tools have come to expect interaction mechanisms such as click-to-select and objects with draggable anchors. In addition, Metoyer *et al.* found that people often treat white space as a manipulable element [177]. Charticulator, as mentioned above, enables chart authors to directly manipulate layout parameters such as anchors, margins, and gaps. On the other hand, many layout aspects cannot be easily expressed as manipulable objects. For example, glyphs can be stacked horizontally or vertically, or they can be laid out in a grid. Since a layout is specified as a combination of partial specifications, we design a concise set of menus and panels to represent such options that cannot be directly manipulated.

5.2.2 Framework

Conceptual frameworks play a central role in visualization authoring tools. For example, the underlying framework of Lyra is the Vega specification [162], and thus Lyra is largely designed around the structure of Vega. Data Illustrator uses a framework based on partition and repetition, and its user interface design is deeply tied to these concepts. Charticulator's framework is designed to support easy creation and composition of novel layouts. Below is the formal specification of the framework, and Figure 5.3 illustrates the use of the framework.

Mark := Rectangle | Symbol | Line | Text

GlyphElement := Mark | Guide | GuideCoordinator | DataDrivenGuide

Glyph := GlyphElement*, LayoutConstraint<GlyphElement>*

ChartElement := PlotSegment | Link | Mark | Legend | Guide | GuideCoordinator

PlotSegment := Glyph, (Scaffold | Axis){0..2}, Sublayout, CoordinateSystem

Attribute := “x1” | “y1” | “x2” | “y2” | …

ElementAttribute<ElementType> := ElementType, Attribute

ParentAttribute := Attribute

ConstraintType := “equals”

LayoutConstraint<ElementType> :=

(ParentAttribute | ElementAttribute<ElementType>){2}, ConstraintType

Chart := ChartElement*, Scale*, LayoutConstraint<ChartElement>*

Notation

“*”: zero or more; “{0..2}”: zero to two; “|”: or;

“X<Type>”: template with parameter “Type”

Layout Elements

A set of elements can be used either at the glyph level or at the chart level. *Marks* are primitive graphical elements (*e.g.*, rectangle, text) whose attributes such as width, height, and color can be manually specified or bound to data. *Guides* are indicators that are visible only during the design phase and are used for aligning objects (*e.g.*, a horizontal guide for aligning the top of marks). *Guide Coordinators* add multiple guides with a layout relationship (*e.g.*, five guides with an equal distribution) to a glyph or a chart.

The *Glyph-level* specification includes *glyph elements* and *layout constraints*, which specify position relationships between glyph elements. Charticulator currently supports only equality constraints. For example, a star symbol can be positioned at the center of a rectangle (*i.e.*, $star.x = rect.cx$, $star.y = rect.cy$). To determine the mark position by data, Charticulator incorporates the *Data-Driven Guides* technique [113]. A data-driven guide provides data-driven anchor points from data columns with the same unit (*e.g.*, min and max values of temperature). Glyph elements can be snapped to these anchor points by

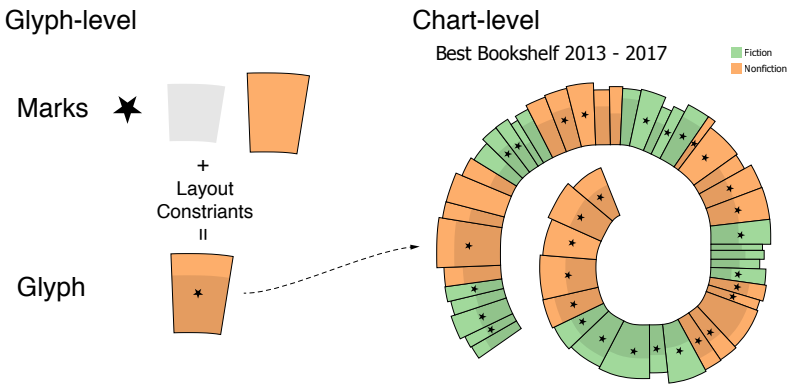


Figure 5.3: An example illustrating Charticulator's framework. Inspired by the *Best Bookshelf* visualization [172], this example uses three marks to compose the glyph: a star symbol and two rectangles. The glyphs are laid out by the plot segment with a horizontal scaffold and a custom curve coordinate system, which morphs the glyphs into wedge-like shapes. A text mark is used for the chart title and a legend shows the color scale.

adding layout constraints (*e.g.*, put the star symbol at the anchor point of the "min" data column). In addition, data-driven guides can be displayed as axes; to avoid duplicated axes, Charticulator shows only the first instance. This is useful for incorporating familiar graphical elements like error bars (*e.g.*, the X-axis in Figure 5.10).

The *Chart-level* specification includes *chart elements*, layout constraints between them, and *scales*. The most important chart element is a *plot segment*, which lays out glyphs according to its *scaffolds* and/or *axes*, and transforms them according to its *coordinate system*, which we discuss in Section 5.2.2. Scales specify how data is mapped to attributes such as width, height, and color, and they can be shared among several marks. *Legends* visualize the scales used in the chart. Charticulator currently uses a predefined legend for each scale type. Links draw between-glyph connections as described in Section 5.2.2.

Layout Composition

The layout of a plot segment is determined by its *scaffolds* and *axes*, and further determined by *sub-layout* options, if applicable. A plot segment has two independent layout directions. The terms we use for these directions depend on the plot segment's coordinate system: "X" (or "Horizontal") and "Y" (or "Vertical") for Cartesian coordinates, "Angular" and "Radial" for Polar coordinates, and "Tangent" and "Normal" for coordinates along custom curves. Similar to Transmogrifier [178], Charticulator morphs mark shapes in non-Cartesian systems. However, while Transmogrifier's morphing is image-based, Charticulator's morphing is performed in vector graphics and thus maintains a precise data binding. For example, a rectangle becomes a wedge shape in a custom curve coordinate system (Figure 5.3).

The X and Y direction of a plot segment can be assigned as a *scaffold*, a categorical *axis*, or a numerical *axis*. A scaffold stacks the glyphs sequentially within the plot segment; a categorical axis groups the glyphs according to their categories to place them with even spacing along the axis; and a numerical axis positions the glyphs based on their data values. Figure 5.4 shows the layouts produced from the combinations of scaffolds and axes.

For categorical axes, Charticulator employs *sub-layouts* to determine a within-group layout. Charticulator currently supports four such options: horizontal stacking, vertical stacking, grid, and circle-packing. However, when a categorical axis is combined with a scaffold or a numerical axis, the sub-layout does not apply because a scaffold or a numerical axis positions glyphs based on their data values (*e.g.*, Figure 5.2-c). When no scaffold or axis is specified, we treat all glyphs as a single group, and thus the selected sub-layout option constitutes the primary layout (*e.g.*, Figure 5.2-f).

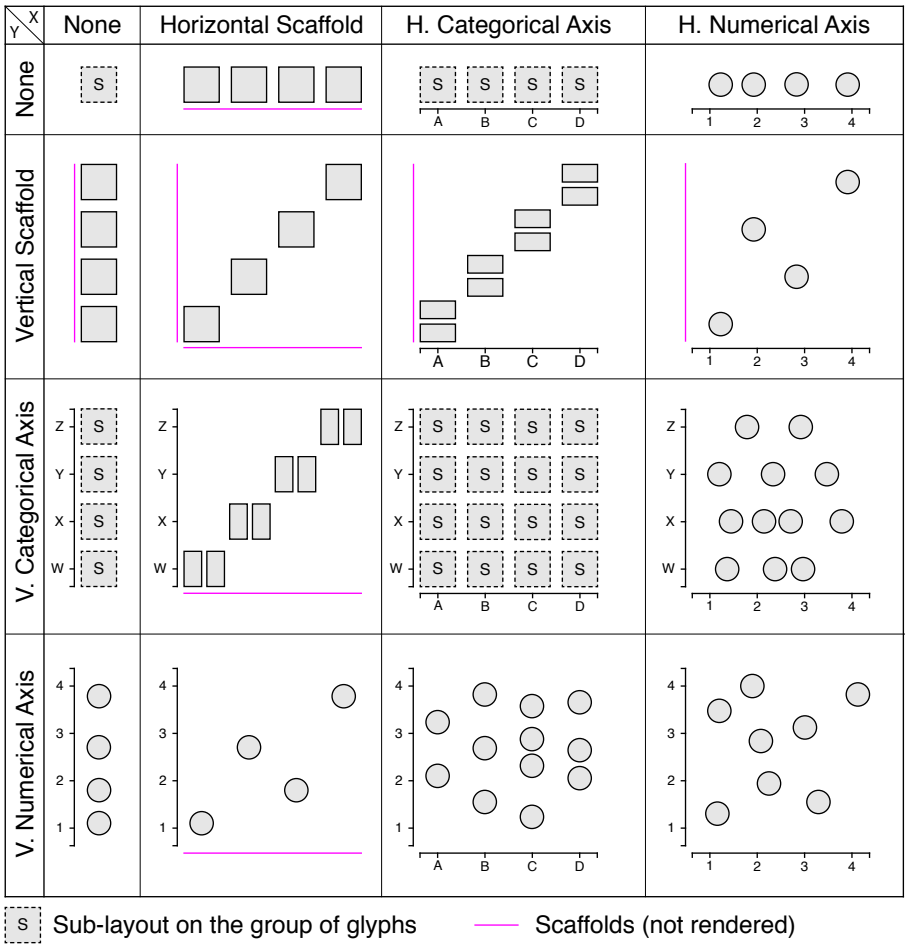


Figure 5.4: Possible combinations of scaffolds and axes.

Link Construction

Charticulator supports three types of *links* for displaying connections between data items: (1) links through a data series (*e.g.*, the link through each Operating System in Figure 5.2-l); (2) links connecting data items on multiple plot segments (*e.g.*, the link between two axes in Figure 5.2-h); and (3) links from a separate data table (*e.g.*, the links for the edges in *Les Misérables*'s character co-occurrence dataset in Figure 5.2-f).

Links must be anchored on glyphs, and can take the form of lines or bands. The shape of links can be straight or curved (Bézier curve, arc). For example, the straight bands

are anchored from the right side of the starting rectangle to the left side of the ending rectangle in Figure 5.2-l, while the line arcs are anchored at the top of each rectangle in Figure 5.2-f.

5.2.3 User Interface and Interaction

Charticulator's user interface consists of a dataset panel, a glyph editor, a layers panel, an attributes panel, and a chart canvas (Figure 5.1). We explain Charticulator's interaction with two example scenarios. To see how Charticulator works in action, refer to gallery videos on the website (`https://charticulator.com`).

Basic Interaction Mechanisms

Using Charticulator, the chart creator can compose a glyph in the glyph editor, adding marks and specifying relationships between them. They can specify the layout relationships between the glyphs either within the chart canvas or from the attributes panel, which displays attributes for the currently selected item.

To add a mark to a glyph, the chart creator can simply drag the desired mark and drop it into the glyph editor. Charticulator places the mark at its default position, the center of the glyph, and adds it to the layers panel. To place the mark at a specific location, the creator can click the mark to activate it and click or drag on the canvas depending on the mark type. Guides and plot segments share the same mechanism.

In addition to selecting it from the layers panel, the chart creator can select a mark from the glyph editor or the chart canvas. For the selected item, Charticulator shows anchors and/or handles, which can be used to specify the layout relationship between two objects in the corresponding canvas. For example, the chart creator can ensure the same gap between a text mark and a rectangle by dragging the anchor of the text to the

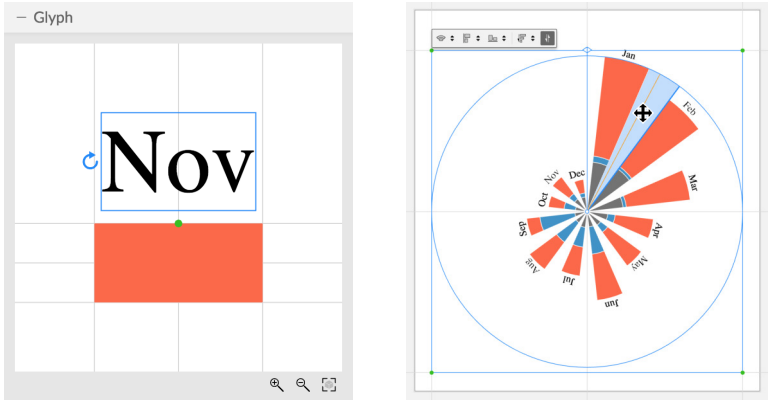


Figure 5.5: Anchors and handles in the glyph editor (left) and in the chart canvas (right). The green (dot) represents that the anchor is snapped to an underlying guide.

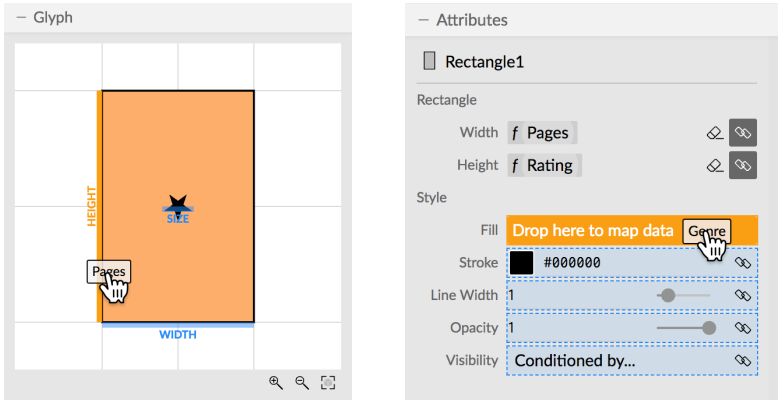


Figure 5.6: Charticulator uses Dropzones [8] for drag-and-drop style data mapping in the glyph editor (left) and the attributes panel (right).

top of the rectangle (Figure 5.5-left). Similarly, they can drag a gap handle to adjust the gaps between glyphs (Figure 5.5-right).

The chart creator can perform data binding in multiple ways. They can drag a data column and drop it into dropzones [8] in the glyph editor (Figure 5.6-left), the chart canvas, or the attributes panel (Figure 5.6-right). They can also select the data column from a popup panel. When the data is bound to mark attributes, Charticulator automatically infers the appropriate scales (categorical and numerical), which can be manually adjusted if necessary in the scale editor (Figure 5.7-left).

The layout of a plot segment can be specified in multiple ways. A plot segment pro-

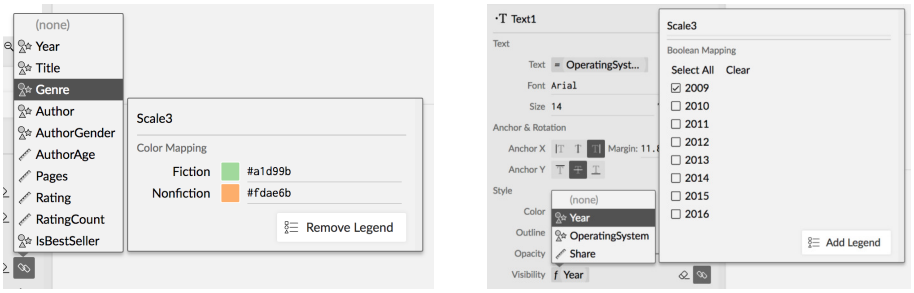


Figure 5.7: Specifying data mapping in the attributes panel. (left) Color-coding marks with `Genre` and editing the color palette. (right) Conditional visibility of marks, showing only the first `Year`.

vides dropzones for binding data to its principal directions such as X and Y for Cartesian coordinates in the chart canvas and in the attributes panel. Once a data attribute is dropped, an axis is created for the direction. To add a scaffold to a plot segment or to change the plot segment's coordinate system, the chart creator can drag a scaffold button from the toolbar to the plot segment's dropzones.

For creating links, the chart creator can specify the shape (*i.e.*, line or band) and type of links from a popup panel (see Figure 5.8), and adjust anchor positions directly from the chart canvas. Charticulator also supports *data-driven visibility*; as the name implies, the visibility of marks is determined by data values. Clicking on the "Conditioned by" button for the Visibility attribute invokes a popup panel, where the chart creator can set a filter via a set of checkboxes (Figure 5.7-right).

As constraints are independently specified, Charticulator supports a flexible order of operations. For example, after specifying the layout of a plot segment, the chart creator can alter the glyph design by adding and removing marks, or by changing layout constraints in the glyph editor; it is also possible to change the glyph layout after adding links.

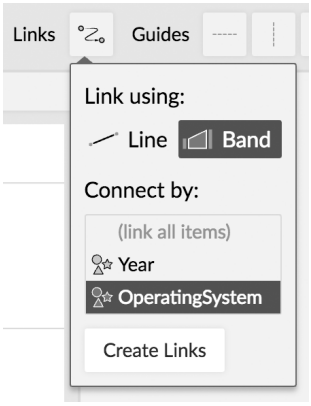


Figure 5.8: Popup panel for link creation

Scenario 1: Mobile Operating System Market Share

We will create a chart depicting the yearly market share trends of mobile operating systems from 2009 to 2016, as shown in Figure 5.2-l. The dataset contains three columns (`Year`, `OperatingSystem`, and `Share`) and 64 rows. In this chart, the values for each year are shown as a stacked column, ordered by the `Share` values. Each operating system is connected by a band; a crossing of the bands shows a rank change.

We begin by adding a rectangle mark to the glyph by dragging it from the toolbar into the glyph editor. To indicate the operating system, we drag the `OperatingSystem` data column to the fill attribute of the rectangle. Then, we bind `Share` to the height of the rectangle by dragging the data column to the height dropzone.

Next, we drag the `Year` column to the X-axis in the chart canvas to group glyphs by year. To stack the glyphs vertically, we toggle the "Stack Y" sub-layout option of the plot segment. To order the glyphs by market share value, we select the `Share` column among the order option for the plot segment. We then adjust the gaps between stacked glyphs and between years by dragging horizontal and vertical gap handles. To create bands between the glyphs, we invoke the "Link" options, select a "Band" link style, group by `OperatingSystem`, and click "Create Links."

To add the text labels, we add a text mark to the glyph by dragging it into the glyph editor, and then we move the text anchor to the middle left of the rectangle and move the text to the left side of the anchor. We then drag the `OperatingSystem` to the Color attribute of the text to make it consistent with the rectangle fill color, and we also drag it to the text dropzone so it is obvious which operating system the glyph refers to. To show the text marks for the first year only (*i.e.*, 2009), we toggle the "Conditioned by" options for the text marks' "Visibility" attribute. From this menu, we select the `Year` column, clear the default selection, and select 2009 only. Finally, we adjust the left margin of the chart by dragging the guide and then type the chart title.

Scenario 2: Reproducing the Rose Chart

We will reproduce Florence Nightingale's Rose Chart, one depicting the monthly death toll and causes of death by disease, wounds, and other causes among the Army of the East during the Crimean War in 1855. In this chart (Figure 5.2-i), each wedge of the circle corresponds to a month, with the three causes of death stacked radially and the area of each being proportional to the number of deaths.

We start by dragging a rectangle mark into the glyph editor. We drag the `Death` column to its Height attribute and the `Type` column to its Fill attribute. To convert Cartesian coordinates to Polar coordinates, we drag the circle scaffold from the toolbar into the chart canvas. This converts the chart into a circular layout of rectangles tangential to the circle. We then drag the `Month` column to the angular axis.

Selecting the plot segment, we toggle a radial stacking sub-layout. We reduce the inner radius of the plot segment by dragging the inner circle to the center (a radius of 0). To bind data (the number of deaths) to the area, we check the "Height to Area" checkbox. To remove angular separation between the wedges, we set the angular gap to 0%, but we give each a white stroke in the rectangle's attributes panel. Next, we select the rectangle

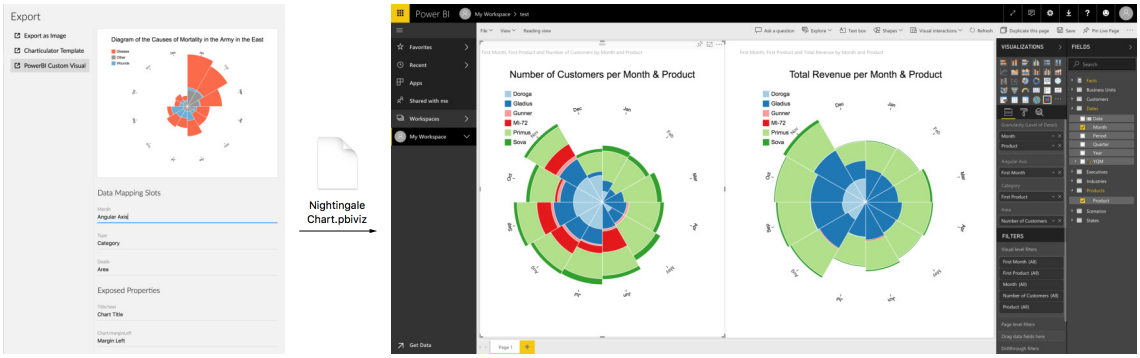


Figure 5.9: Charticulator allows the chart creator to export chart layouts as reusable templates in the form of custom visuals for Microsoft Power BI, which can be imported into Power BI's web or desktop client to visualize new data.

mark to assign a custom palette of colors to the causes of death, and we toggle a legend. Finally, we adjust the chart margins, anchor the legend to left guide, and edit the chart title.

Since this is a chart layout that we would like to reuse, we export it as a Power BI custom visual [179]. We open the File panel, toggle the Export tab, and select "Power BI Custom Visual." In the following panel, we replace the data column names with more generic data slot names: `Month`, `Type`, and `Death` are replaced by "Angular Axis," "Category," and "Area," respectively. We name this template "Rose Chart" and click the "Export" button. Finally, we import the resulting ".pbiviz" file in Power BI and use it to visualize other data (*e.g.*, Figure 5.9).

5.2.4 Constraint-based Layout

At the system level, partial layout specifications are not independent of one another. For example, binding data to the width of a rectangle and stacking all rectangles horizontally to a total width are two inter-dependent partial specifications: the scaling parameter of the width binding depends on the total width of the stacking layout. To support the combination of partial layout specifications, we express partial specifications as mathe-

matical constraints and employ a constraint solver to determine the final layout.

Constraints at Multiple Levels

Since there are chart-level elements, glyph-level elements, and scales to be considered, Charticulator generates constraints at multiple levels.

Chart-level constraints specify how plot segments, legends, chart-level marks, and guides are related to one another. For example, a constraint that legend A should be on the right of the plot segment B translates to $B.x_2 = A.x_1$ mathematically. Guides and guide coordinators can be used at this level to position elements.

Each plot segment has constraints specifying how its glyphs are positioned, and these constraints depend on the plot segment's scaffolds. For example, a single horizontal scaffold enforces the glyphs to be horizontally adjacent to one another (with an optional gap):

$$\forall G_1, G_2 : Glyph \wedge adjacent(G_1, G_2), G_1.x_2 + gap = G_2.x_1$$

$$G_{first}.x_1 = PlotSegment.x_1$$

$$G_{last}.x_2 = PlotSegment.x_2$$

The two preceding statements are boundary conditions, while the following vertically aligns glyphs by their anchors:

$$\forall G : Glyph, G.anchor.y = y$$

Glyph-level constraints are similar to chart-level constraints except that their scopes are individual glyphs. Guides, guide coordinators, and data-driven guides [113] can be used at this level.

A mark or chart element can have intrinsic relationships among its attributes. For example, consider the attributes of a rectangle mark: x_1, x_2, x_{center}, and $width$. Increasing x_2 will also increase $width$ and x_{center}. Intrinsic constraints enforce these relations. In this example they are:

$$x_1 + x_2 = 2 \times x_{center}$$

$$x_2 - x_1 = width$$

Intrinsic constraints allow us to expose multiple related attributes simultaneously and maintain the consistency of partial specifications. Chart authors do not have to fully specify all of the attributes. In the above example, they can specify any two of the four attributes, and the constraint solver computes the remaining attributes.

Data Binding Constraints

Constraints are also involved in the data binding process when layout attributes are bound to data. For example, when a data dimension is bound to the width of a glyph, the scale constraint will be:

$$\forall i, G_i.width = Scale.factor \times d_i$$

Assuming a horizontal layout which enforces the rectangles to be adjacent to each

other, we will have the following constraints:

$$\forall i, G_i.width = Scale.factor \times d_i$$

$$\forall i, G_i.x_2 - G_i.x_1 = G_i.width$$

$$\forall j = i + 1, G_i.x_2 + gap = G_j.x_1$$

$$G_1.x_1 = PlotSegment.x_1$$

$$G_N.x_2 = PlotSegment.x_2$$

The first statement is the scale constraint, the second statement is the glyph's intrinsic constraint, and the third to fifth statements refer to the plot segment's layout constraint. Together, the constraint solver can determine the correct scale factor:

$$Scale.factor = \frac{PlotSegment.x_2 - PlotSegment.x_1 - (N-1)gap}{\sum_i d_i}$$

For the sake of simplicity, we are assuming that the plot segment's attributes are given. In Charticulator, these attributes are subject to chart-level constraints: plot segments can be snapped to other chart-level elements, such as plot segments and guides.

Two-stage Constraint Solving

The mathematical constraint solver we are using is a linear solver, which is fast and has guaranteed convergence. However, many possible charts, such as those incorporating a packing layout or a force-directed layout, can only be expressed via nonlinear constraints. Charticulator therefore incorporates a two-stage constraint solving mechanism. In the first stage, Charticulator invokes the linear constraint solver to determine the layout of the chart elements and a basic binding for stacking-based layouts, which can be expressed via linear constraints. The first stage can generate additional variables that can

be used in the second stage. Then, Charticulator invokes additional layout algorithms (*e.g.*, circle packing) to address nonlinear layouts.

5.2.5 Reusable Chart Templates

A notable strength of Charticulator is that the chart design is not tightly coupled with the input data because automatic layout computation allows the chart to adapt to new data as long as the types of data columns are the same. This makes it possible to export the chart design as a template to be used in other charting systems. Charticulator extracts a list of data mapping slots (*e.g.*, "X-Axis," "Group By") and properties (*e.g.*, "Chart Title") from the chart specification, and allows the chart creator to choose names for the slots and properties (*e.g.*, Figure 5.9-left). Once the custom names are specified, Charticulator bundles the named list with the chart specification and essential Charticulator components including the constraint solver and the chart renderer into a single JavaScript package. This package provides an API to render the chart design with given data columns and properties. For each target visualization tool, we implement an adapter that exposes the API in the format required by the visualization tool. For example, to export as a Microsoft Power BI custom visual, we produce a ".pbiviz" file with appropriate metadata (Power BI refers to these as "capabilities") and glue code. The metadata tells Power BI what the input data should be like, and the glue code takes the input data from Power BI and produces the chart using the exported JavaScript package.

5.2.6 Implementation

Charticulator is implemented as an HTML5 application and uses technologies including TypeScript [180], React [181], and WebAssembly [182]. It follows the basic Flux application architecture [183], with the addition of a constraint solver. The application

maintains a *chart specification* and a *chart state*. The chart specification describes the chart in a JSON object that mirrors the framework discussed in Section 5.2.2. The chart state stores all of chart elements' attributes. Together, they form the "Store" part of the Flux architecture. For each chart editing interaction, an "Action" is emitted and dispatched through the global "Dispatcher" to the "Store." The store then modifies the chart specification and invokes the *constraint solver* component to update the chart state. Once the state is successfully updated, the "Store" emits an update event which causes the user interface components to update.

Many algorithms in the literature solve linear constraints. Notably, the Cassowary algorithm [184] extends the simplex method to allow for prioritizing constraints, and the Conjugate Gradient algorithm [185] is very efficient for solving sparse linear systems. We experimented with both algorithms and settled on a sparse least-squares conjugate gradient algorithm. We selected a sparse solver because most layout constraints involve only a few variables and thus produce a very sparse matrix; we use the least squares minimization technique to deal with weighted constraints, if any. However, this algorithm only supports equality constraints, and thus Charticulator only allows for the specification of such constraints.

More details about implementation as well as algorithm benchmarks can be found in the supplemental material.

5.3 Evaluation

We evaluate Charticulator in three forms: (1) a gallery to demonstrate its expressiveness; (2) a user study to assess its usability; and (3) a click-count comparison with three existing chart creation tools.

5.3.1 Gallery

To demonstrate the expressive power of Charticulator, we produced a variety of charts with a diverse collection of datasets; Figure 5.2 shows 12 examples from our gallery. Many of these charts are reproductions or variations of charts created by news graphics teams. The full gallery can be found in the website, along with high-resolution images, detailed descriptions, and videos illustrating the creation processes.

5.3.2 User Study: Chart Reproduction

To determine if people can produce bespoke charts with Charticulator, we followed a procedure similar to those used to evaluate Data Illustrator [67] and ChartAccent [11], which focuses on chart reproduction.

Study Design

We recruited 11 participants (5 female) who had normal or corrected-to-normal vision from the Puget Sound area. All participants had at least three years of experience using graphics editors such as Adobe Illustrator, Adobe Experience Design, Sketch, and Inkscape. They also had created infographics and used spreadsheets (*e.g.*, Microsoft Excel, Google sheets, Apple Numbers) within the past three months. In addition to five designers, we had participants with six other occupations: a print operator, an architect, a digital marketing manager, an editor, a sales & marketing professional, and a major gift officer. Their ages ranged from 22 to 48, with an average age of 33. We compensated them with a $150 eBay gift card.

We prepared four chart reproduction tasks, covering the basic concepts of Charticulator. Each subsequent task increased in terms of complexity. Task 1's chart depicts monthly activity by intensity in a stacked radial layout (similar to Nightingale's rose

chart) using the activity tracking data. Task 2's chart (Figure 5.2-k) depicts the 200 types of mushrooms grouped by odor and surface quality, using a circle packing sub-layout. Task 3's chart depicts the co-occurrence of characters in *Les Misérables* using a radial layout similar to the chart in Figure 5.1. Task 4's chart (Figure 5.2-l) is described in Section 5.2.3. Videos illustrating the step-by-step construction of these charts are provided on the website. We also prepared six additional charts, three for a tutorial and three for practice tasks.

We used a 3.47 GHz Windows 10 desktop machine with 12 GB RAM, using two side-by-side 24-inch LCD displays running at 1920×1200 resolution. For each task, we showed the target chart image on the left monitor, and asked participants to reproduce the same chart in Charticulator on the right monitor. We logged participants' interactions with Charticulator and recorded chart images at each construction step. We also captured screen recordings along with concurrent video and audio recordings of the participants as they proceeded.

After providing a brief explanation of the study goals and procedure, we asked participants to complete a pre-study background questionnaire. We then provided a tutorial on the basic features of Charticulator using three charts. After completing three practice tasks to familiarize themselves with Charticulator, participants performed the four tasks described above on their own. Before starting each task, we described the target chart and the underlying dataset. When they are ready, participants pressed a "Start Task" button to load the dataset and begin the task. After completing the task, they pressed a "Complete Task" button. We encouraged participants to think aloud, especially when any aspect of the task was unclear, or if they were unsure of how to use Charticulator's features. We provided hints to participants when they asked for help or when their progress stalled, noting the cause in such cases. At the end of the study session, participants filled out a questionnaire about their experience with Charticulator. On average, the train-

ing (tutorial + practice) lasted about 50 minutes, and the entire session lasted about 80 minutes.

Results

Ten out of 11 participants successfully reproduced the target charts for all four tasks with only a few hints, and six participants successfully completed all tasks without any hints. The one remaining participant (P3) completed the first three tasks, but had trouble understanding that, in Task 4, the bars were ordered by the share values in each year.

We conducted two pilot sessions before this study, and in doing so we identified three usability issues: (1) text manipulation (*i.e.*, alignment and rotation), (2) conditional visibility (of text), and (3) legend creation. To address these issues, we incorporated direct manipulation for the text mark and revised the attributes panel. Our results indicated that we fully addressed the first two issues. None of the participants had trouble manipulating text marks. All participants managed to control the visibility of text marks. However, two participants still forgot how to add a legend. Considering a legend as a property of the chart itself, they tried to look for a button to add a legend at the chart level. P9, who forgot how to add a legend during the practice session, mentioned that *"I'm still thinking in a regular [Microsoft] Excel format."*

With regards to task completion time, the average task completion time was less than four minutes, ranging from 130 to 235 seconds (Figure 5.10). Note that we did not explicitly ask the participants to complete the task as quickly as possible. When rating Charticulator on three satisfaction criteria using a 7-point Likert scale (1: "Strongly Disagree" to 7: "Strongly Agree"), participants indicated that Charticulator is easy to learn (Avg = 6.55) and that it was easy (6.27) and enjoyable (6.73) to create charts with it. Participants appreciated various aspects of Charticulator. Seven participants mentioned that the drag-and-drop interaction was what they liked most. Participants also appreciated

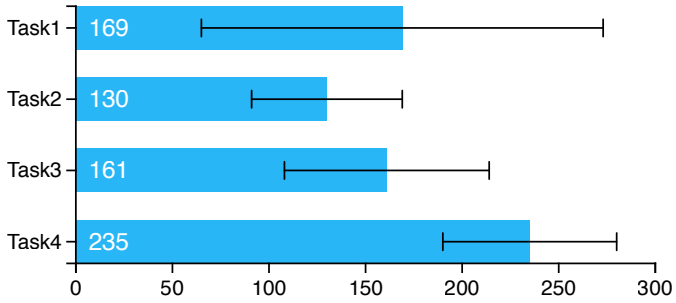


Figure 5.10: Task completion time in seconds. Error bars represent the standard deviations. (This chart is created with Charticulator.)

the immediate visual feedback, its power and flexibility, its similarity to Adobe InDesign, and its easy access to the underlying data.

5.3.3 Comparison to Existing Tools

We also sought a way to compare Charticulator with three existing chart creation tools: Data Illustrator [67], Lyra [8], and iVisDesigner [10]. While a comparative user study would have been ideal, these tools differ in terms of conceptual framework, feature set (*e.g.*, z-ordering, undo/redo), the availability of tutorials, low-level interface design choices, and the stability of their implementation. We thus decided to compare the number of user interactions as a proxy assessment of their respective complexities, inspired by the keystroke-level model [186] approach. However, we concede that a fewer number of interactions does not necessarily mean that a tool is easier to learn to use the tool.

We proceeded to generate seven charts (Table 5.1) with each of the aforementioned tools. We selected charts that are representative of a variety of designs and layouts, those that can be created with other tools. We considered the chart creation to be successful if it was possible to realize the chart design with reasonable fidelity, focusing on the correctness of the data binding, layout, and linking because exact chart size, legend, label positioning, and colors can vary given a tool's available features. In addition, we did not

count repetitive interactions to fine-tune a design (*e.g.*, trying out different colors or gap sizes), and assumed that one such interaction yielded the desired result.

We report the number of clicks, drag-and-drops, and text inputs separately (Table 5.1). Double-clicks count as two clicks; one text input interaction includes a click to activate the textbox. The actual counts may vary depending on the exact order of creation. We again note that these numbers should only be treated as a general impression of the complexity of interaction. Based on this comparison, we found that Charticulator has a comparable complexity to other tools in terms of the number of low-level interactions.

5.4 Discussion and Future Work

5.4.1 Limitations

Framework Charticulator currently cannot create three charts from Data Illustrator's gallery. While some of the limitations can be addressed by simply implementing more features (*e.g.*, adding a triangle mark type), other limitations are inherent to the framework design. The most notable limitation is that Charticulator supports only one level of repetition in plot segments, and thus the data granularity must be predetermined by the chart creator. In contrast, the desired granularity can be achieved by multiple repetition and partition operations in Data Illustrator's framework. To address this limitation, we would like to extend our framework in two ways. First, a data transformation pipeline can be added to the plot segment level; by specifying a set of data transformations, the chart creator will be able to aggregate the data to a desired granularity. Second, and more importantly, we can allow a glyph design to be a nested chart (*e.g.*, Figure 5.2-e can be seen as a scatterplot of bar chart glyphs): this can be done recursively to support multiple levels of granularity.

	Charticulator			**Data Illustrator** [67]			**Lyra** [8]			**iVisDesigner** [10]		
Chart	CL	DD	TI	CL	DD	TI	CL	DD	TI	CL	DD	TI
Nightingale chart (Figure 5.2-i)	31	13	2	Not supported			24	16	7	Not supported		
Color-coded matrix [a] (Figure 5.2-b)	15	13	2	20	5	2	11	12	2	27	1	2
Parallel coordinates [b] (Figure 5.2-h)	18	10	5	20	5	1	29	9	9	26	0	5
Red and Blue America [187]	20	8	2	43	9	1	14	9	2 [c]	57	4	4
Ranking of CO_2 Emissions (Figure 5.2-c)	29	6	3	44	3	2	45	11	3	60	0	11
Caltrain's schedule [d] (Figure 5.2-g)	13	8	4	39	9	2	22	16	5	47	1	4
Best bookshelf (Figure 5.2-j)	26	11	2	50	8	1 [e]	36	19	7 [e]	Not supported		

Table 5.1: Number of clicks (CL), drag-and-drops (DD), and text inputs (TI) to create each chart using Charticulator and three existing tools. Notes: (a) While the original chart has a multi-stop custom gradient, we opted to use a two-stop gradient for the sake of simplicity; (b) Axis labels are omitted and circles are optional; (c) We were unable to add state name text labels, but this appears to be a bug and not a shortcoming of the tool; (d) The Y axis represents the distance between the stations; and (e) We were unable to determine how to add the stars to indicate bestsellers. Lyra supports connectors, but it does not work as desired with a stacked layout.

Scalability The main drawback of incorporating a constraint solving algorithm is speed (and thus scalability), as most of the constraint solvers have $O(n^2)$ time complexity. For example, positioning a few hundred glyphs with chaining constraints (*e.g.*, stacking them horizontally with varying widths) may take 1–2 seconds to complete. We have improved the solving speed around 1.5–2 times by using WebAssembly [182] and running the solver in a background worker to avoid blocking the user interface. In the future, we will investigate ways to further improve the performance of the constraint solving algorithm, for example, by using a preconditioner such as incomplete Cholesky factorization [188] in the conjugate gradient algorithm to address chaining constraints.

User study In our reproduction study, we intended to assess if people could produce chart layouts using Charticulator when provided with a reference chart, data, and about an hour of training. However, just as it is difficult to ascertain if a particular text editor will result in highly expressive prose, we cannot conclusively determine if the use of Charticulator will result in the production of expressive and novel charts. We invested a significant amount of time and effort to design bespoke charts for our gallery and for our reproduction study, and thus it would be logistically difficult to study the design of truly bespoke chart layouts in 60–90 minute laboratory study sessions. In the future, we plan to evaluate the expressiveness of Charticulator with chart creators in a workshop or hackathon setting. To promote engagement with the tool, we will ask participants to bring their own data. We also intend to conduct a long-term evaluation through engagement with students taking a data journalism course taught by our academic collaborators.

5.4.2 Manual Manipulation vs. Layout Specification

People familiar with vector graphics editing tools such as Adobe Illustrator are familiar with the "everything is manipulable" idea and have come to expect that objects

will respond to direct manipulation. In a constraint-based layout approach, this may not always be feasible because constraints are inherently interconnected, and chart creators can try to set conflicting constraints. To alleviate this issue, Charticulator controlled the complexity of constraints that can be specified, in an attempt to minimize the level of unexpectedness. It would be important to investigate ways to provide appropriate feedback to chart creators when their desired layout is not possible due to conflicting constraints.

Visualization design generally involves specifying a large number of attributes. For example, a rectangle mark has width, height, color, stroke, opacity, and visibility. Furthermore, people tend to iteratively design elements and revisit earlier design choices. Even though our study participants were familiar with graphics editing tools, some participants were overwhelmed by the many options Charticulator offered. For example, P8 stated that *"It was sometimes difficult determining what I needed to click to reveal other properties/options."* In several occasions, knowing that they needed to apply a packing sub-layout, participants tried to remember where the option for the sub-layout was. Thus, an important direction for future research would involve envisioning new forms of interactive tutorials to facilitate the self-teaching of highly configurable user interfaces such as Charticulator's.

5.4.3 Additional Expressivity

Charticulator currently includes only four basic mark types. We can support more complex glyph designs beyond the composition of rectangles, symbols, lines, and text. For instance, incorporating the "Pen" tool from Adobe Illustrator would allow chart creators to draw arbitrary polylines or curves. The control points of these elements can be anchored on other marks or positioned by data-driven guides [113]. Similarly, it would be useful to investigate the use of pen and touch interactions to create custom marks

such as in DataInk [189]. In addition, we can further enhance the expressive power of Charticulator by incorporating the annotation capabilities of ChartAccent [11], which provides a rich palette of data-driven annotation options.

Charticulator primarily deals with layouts of glyphs. Links are connected to glyphs by user-specified anchors, and no further layout of links is supported. In the future, we will investigate how to interactively specify the layout of links by incorporating edge bundling (*e.g.*, hierarchical edge bundling [190]) and routing techniques (*e.g.*, force-directed edge routing [191]), which may involve the insertion of magnetic elements that interact with a force-directed edge routing algorithm to aid the interactive placement of links. Besides, we can further extend Charticulator's expressive range by supporting additional specialized layout algorithms (*e.g.*, treemap and force-directed graph layout).

As demonstrated in Figure 5.9, Charticulator can export charts as reusable templates. Our ultimate goal is to allow Charticulator to *import* these templates and use them as glyph-level elements that can be bound to data. This will facilitate the creation of faceted charts and small multiples. For example, we can import a line chart of monthly market share values and display it at multiple geographical locations.

5.4.4 Deployment: Reaching a Worldwide User Community

Microsoft Power BI is an established business intelligence tool with a large worldwide user community, which includes a marketplace for sharing custom visuals. We are presently collaborating with Power BI to deploy Charticulator as a publicly available custom visual designer. In conjunction with this deployment, they also intend to host user community events such as a custom visual design competition. We are excited about the opportunity to reach Power BI's user community, which will help us better understand how people use Charticulator with their own data.

5.5 Conclusion

In this chapter, we presented Charticulator, an authoring tool for specifying bespoke and reusable chart layouts. Unlike other interactive charting tools, Charticulator prioritizes the configuration of layout between marks or glyphs, and leverages a constraint solver to achieve a wide variety of chart designs. We explained the design principles behind Charticulator's framework, which generalizes to the creation of a wide range of charts. We described how Charticulator transfers the chart specification into layout constraints and incorporates a constraint-based layout algorithm, which enables the export of chart designs as reusable chart templates. We also illustrated how its user interface enables interactive chart layout specification. We demonstrated the expressive potential of Charticulator through a gallery of charts, assessed its usability via a chart reproduction study, and counted the number of interactions required to create charts, comparing against three existing tools. Finally, we discussed several ways to further evaluate and enhance the expressive power of Charticulator.

Charticulator is publicly available at `https://charticulator.com`. The source code of Charticulator is released under the MIT open source license at `https://github.com/Microsoft/charticulator`.

Part II

Visualization Authoring for Immersive Environments

In Part I, we have presented three tools for authoring information visualizations. However, these user interfaces mostly work in traditional windows, icons, menus, pointer (WIMP)-style environments. In this part, we explore visualization authoring for future user interface environments including virtual and augmented reality.

A simple authoring system for immersive visualization We start by presenting a simple extension to iVisDesigner to support visualization authoring in virtual reality. This system was primarily used for conducting a user study to evaluate the effect of field of view (FOV) in wide-field-of-view augmented reality visualizations [9]

Stardust The recent growth of virtual and augmented reality poses a challenge for supporting multiple display environments beyond regular canvases, such as a Head Mounted Display (HMD) and Cave Automatic Virtual Environment (CAVE). Web-based visualization libraries are in wide use, but performance bottlenecks occur when rendering, and especially animating, a large number of graphical marks. While GPU-based rendering can drastically improve performance, that paradigm has a steep learning curve, usually requiring expertise in the computer graphics pipeline and shader programming. In Chapter 7, we introduce a new web-based visualization library called Stardust, which provides a familiar API while leveraging GPU's processing power. Stardust also enables developers to create both 2D and 3D visualizations for diverse display environments using a uniform API. To demonstrate Stardust's expressiveness and portability, we present five example visualizations and a coding playground for four display environments. We also evaluate its performance by comparing it against the standard HTML5 Canvas, D3, and Vega.

Idyll-MR With immersive AR and VR platforms becoming commonplace, interactive immersive data visualization holds great potential for delivering more engaging data-

driven stories. There are currently few solutions for effective authoring of such data-driven stories, and an effective cross-platform collaborative authoring, presentation, and interaction environment would have great applications in data-driven storytelling. In Chapter 8, we present such an environment, Idyll-MR, which supports declarative authoring of immersive data-driven stories. It builds upon and extends the existing Idyll language [192], which was originally designed for authoring of conventional 2D visual data stories. We also present the design and implementation of the Idyll-MR infrastructure and our design choices for immersive data-driven storytelling components. We discuss its capabilities and implications, including detailed examples demonstrating the use of Idyll-MR in three different scenarios.

Chapter 6

A Simple Authoring System for Immersive Visualization

6.1 Introduction

In this chapter we describe a simple immersive visualization authoring system. This system targets the AlloSphere [136], a three-story high spherical projection-based immersive environment. We incorporate iVisDesigner's web-based interface to support visualization construction, and present the user with a tablet-based multi-touch interface (Figure 6.1) to place these visualizations in the spherical surface.

The AlloSphere consists of a five-meter radius spherical surface, within which a two-meter wide bridge. The user is supposed to stand at the center of the bridge to get the best viewing perspective. In our system we use two interaction modalities: a laptop interface running iVisDesigner, and a tablet interface running our visualization placement software. In the following sections we illustrate the system in more detail.

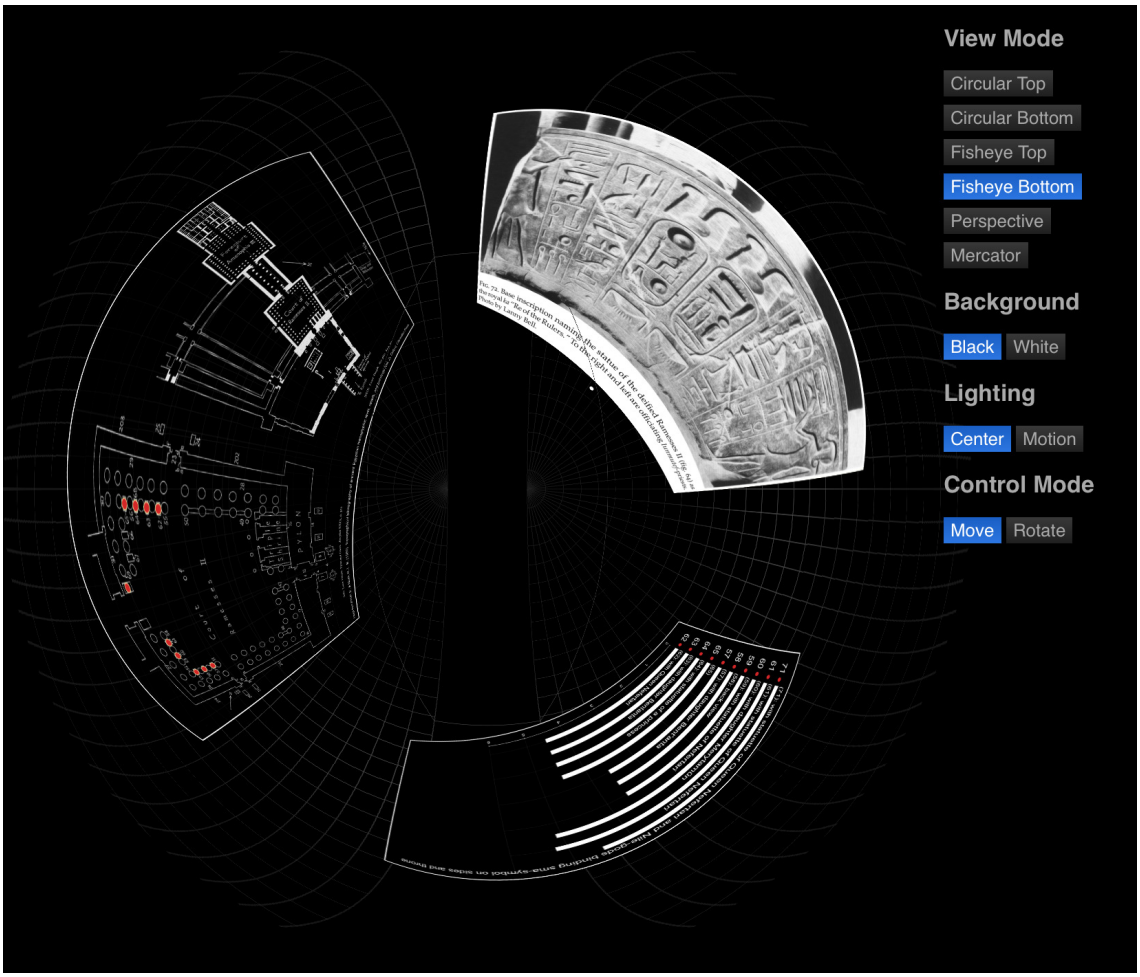


Figure 6.1: The tablet interface for visualization placement.

6.2 Visualization Placement

We introduce a tablet-based interface for visualization placement, as shown in Figure 6.1. It gives the user an overview of the virtual spherical surface on which the various 2D visualizations are presented, and provides standard multi-touch gestures to scale the visualizations and move them around. User actions on the tablet are immediately reflected in the spherical display.

Users can immediately switch among the supported projection modes: circular, fish-

eye, mercator, and perspective. The circular and fisheye projections give the user a global overview; the mercator gives a partial overview with less distortion; the perspective mode reflects what the user sees (using swipes to rotate the limited field of view), making it easier for small adjustments.

We impose some constraints on the visualization placement. Since the user is always standing at or nearby the center of the sphere, we can safely assume the viewing direction and the up vector of these visualizations. Therefore, we restrict them to tangent rectangles of the sphere surface, and always upwards, which means they always face directly to the users on the bridge. These constraints can be relaxed by extra user interaction: the user can rotate the visualizations, which can be useful to understand the links between them, which are in 3D space.

6.3 Visualization Linking

Besides creating 2D visualizations and placing them around in the spherical surface, we also supported drawing 3D links between them.

There are two ways to create links. **Directly Linking Visualization Elements**: Users select a set of data items and pick objects from different visualizations for linking. **Linking by Data Items**: Link all graphical elements that are bound to the selected data items. The system will create links between all pairs of graphics corresponds to the selected data items. Link creation is done via a panel in iVisDesigner's web-based interface, running on the user's laptop. The interface allows editing four link properties: Filter, Color, Thickness, and Curveness. A curved line can either bend towards the user or backwards.

We employed the Illuminated Steamlines [193] approach to render the links, and also highlighted the endpoints of the links for better visibility. The light source can be either fixed at the center or moved around the user at a constant speed.

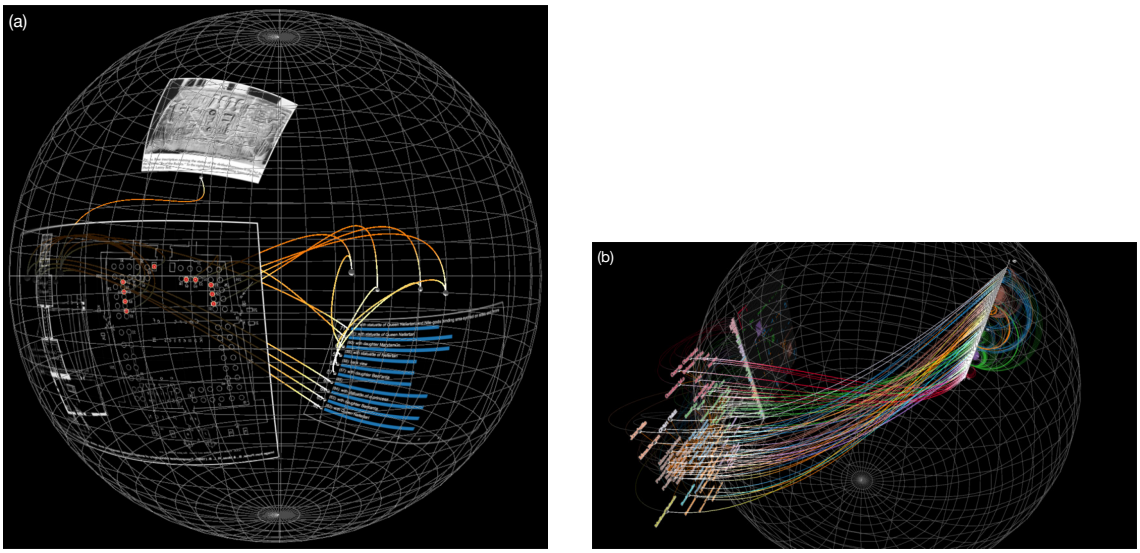


Figure 6.2: Outside views of the 3D virtual space. (a) Hybrid stereo mode: each chart lies on the spherical surface, allowing for zero disparity. (b) Full stereo mode: each chart is a planar surface in the 3D space.

6.4 Integration with Virtual Environments

In a 3D virtual reality setting, 2D visualizations can be displayed as overlays of the environment. Links can be drawn from visualizations to places in the virtual environment as well. In the context of augmented reality, this can be considered as a simulation, where the background environment simulates the real world and the visualizations are what the users would see with an AR device. The simulation of augmented reality experience can be useful for conducting user studies about the visualization designs when an actual AR device is not yet available.

In our prototype, we imported a set of stereo panoramas as a demonstration. While not yet supported in our prototype, it's theoretically possible to use animated stereo panoramas, immersive 3D virtual world or even realtime streamed ones.

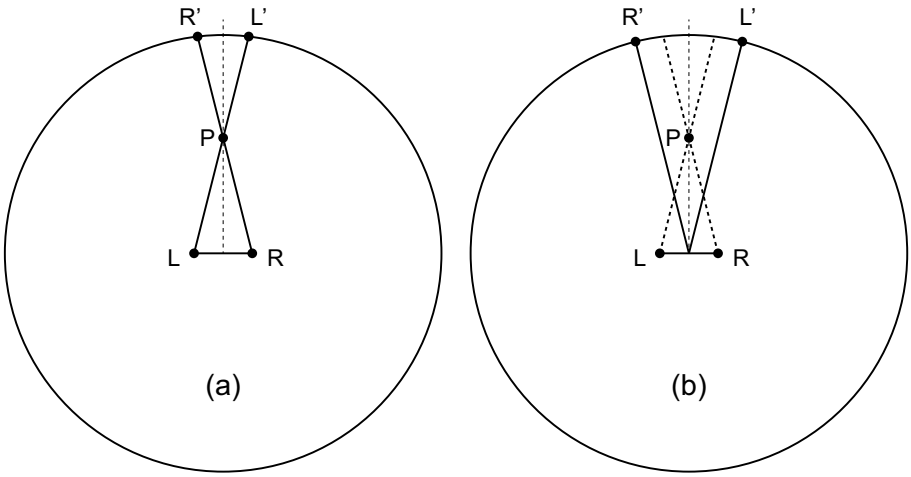


Figure 6.3: OmniStereo rendering methods. (a) Zero parallax at the sphere surface, negative parallax inside, positive parallax outside. (b) Negative parallax, can be viewed as mode (a) with sphere radius far more than eye separation.

6.5 Stereo Modes

Here we discuss our stereoscopic rendering considerations. Since active stereo glasses and anaglyph glasses reduce luminosity by at least a half, and have other issues such as interference with other displays and occluding facial expressions, it will be helpful to enable users to view the scene without stereo glasses. To facilitate this, we present three stereo modes for displaying the visualizations and the links.

6.5.1 Monocular Mode

Monocular mode simply means the scene is rendered without any disparity, which allows users to see things without wearing glasses, but users can't experience stereo at all.

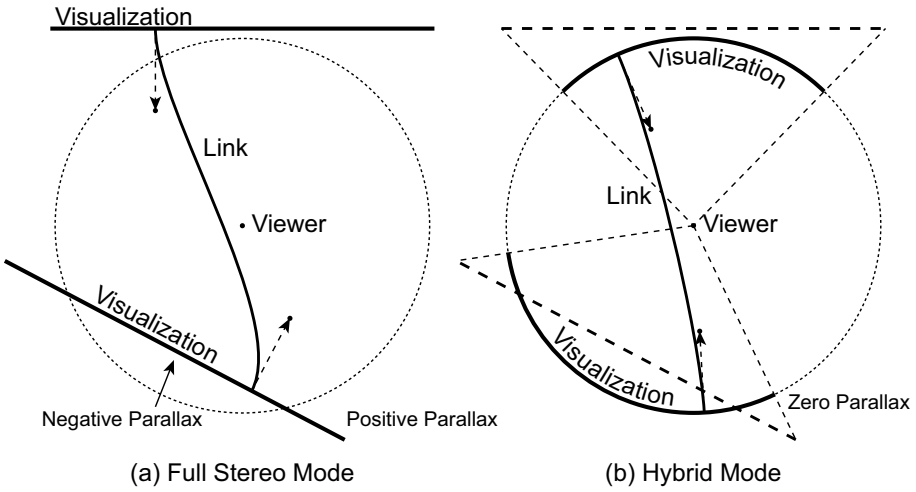


Figure 6.4: Stereo modes. In the hybrid mode, the visualizations are positioned on the screen surface, therefore have no parallax.

6.5.2 Full Stereo Mode

In the full stereo mode, everything is rendered in stereo, therefore in this case users must wear glasses or see double images. This mode gives the best depth perception, because the depth cues given by stereo disparity and perspective correspond well. Figure 6.2 (b) shows how the visualizations are displayed in the 3D space in full stereo mode.

6.5.3 Hybrid Mode

Hybrid mode means the visualizations are rendered without any disparity, but the links are rendered as stereo. This mode allows users to see the visualizations clearly without glasses. Figure 6.2 (a) shows how the charts are located in the 3D virtual space in the hybrid mode.

By carefully adjusting the link endpoints on the visualizations to the distances suggested by zero disparity, and tweaking the depth buffer to allow for correct depth ordering, we were able to get a coherent view in this mode. We support two OmniStereo [194] rendering modes, as shown in Figure 6.3. In the hybrid mode, rendering mode (a) is used.

Since the links between visualizations are rendered with disparity in this mode, users not wearing glasses will experience double images for the links. From our experience, the links can still be perceived correctly even without stereo glasses, because the joints between the links and the visualizations are still without disparity, and the double lines follow the same trend, both of which help reduce confusion. With glasses, users could get much better feeling of depth. The other potential problem is the mismatch between stereo disparity and perspective cues. However, for most visualizations, there are enough vertical or horizontal lines to give a strong perspective cue, therefore users' eyes will believe that they are actually planar objects rather than spherical ones.

In our prototype system, users can switch between these modes as desired. In hybrid mode, users can still change the orientation of the visualizations (by relaxing our constraint that visualizations must be tangent to the sphere surface and upright). In this case, if users tilt the visualizations too much, it will introduce some perceptual problem, because for a tilted visualization, the mismatch between stereo disparity and perspective cues becomes much stronger. Therefore, we suggest that users should use full stereo mode to get the best experience if they want to tilt the visualizations.

6.6 Mixed Reality Simulation

As discussed above, we have designed and implemented a system that allows us to interactively create content in a full-surround display environment (as illustrated in Figure 6.5 and Figure 6.6). We used the system to create the annotations for our experiment.

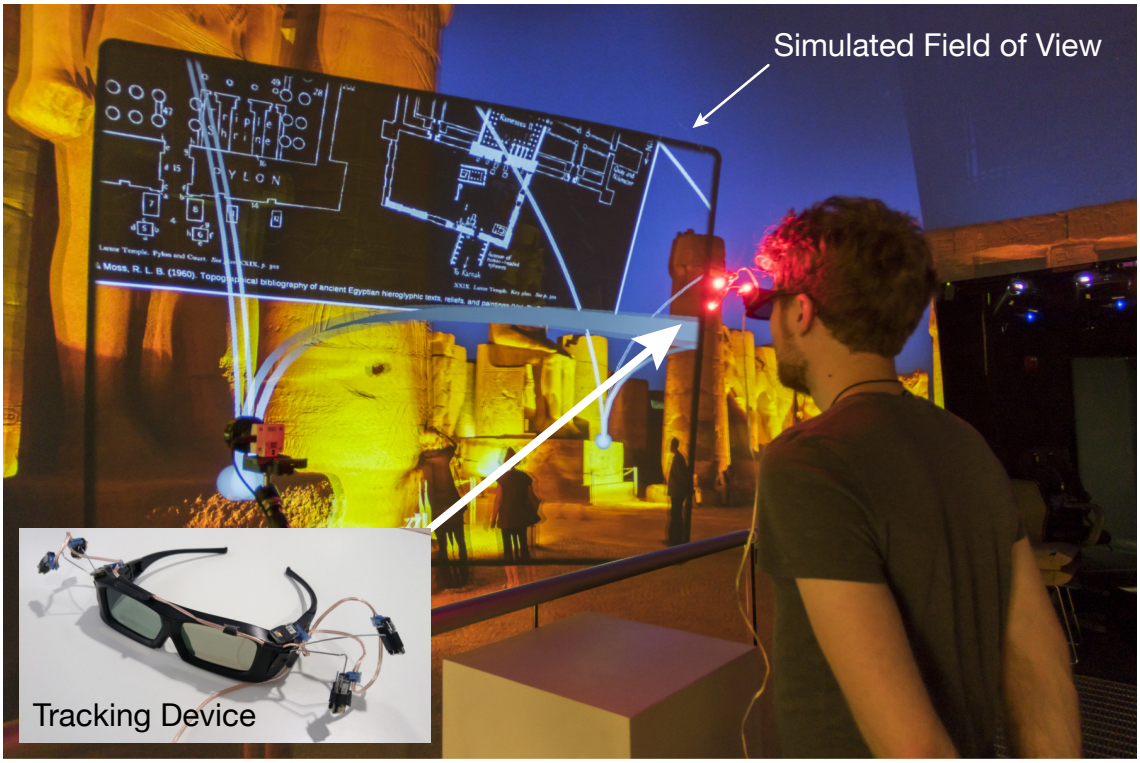


Figure 6.5: Simulating AR with a small field of view. We let the viewport follow the user's head movement. The user only see a small part of the annotations. The border of the viewport simulates the border of the display of an AR device. The stereo panorama of the Luxor Temple is created by Tom DeFanti *et al.* [195].

6.6.1 Simulating Augmented Reality Parameters

We designed an AR simulator for use with the system. It can simulate two important parameters for AR devices: field of view and tracking artifacts.

Simulating field of view Unlike head-mounted AR/VR devices, our display is fixed, surrounding a user standing in the middle of it. Therefore, in order to simulate a small augmentation field of view, we have to know the head orientation with respect to the display. Head tracking was performed by a PhaseSpace Impulse X2 system [196] which uses active optical tracking. We installed 6 PhaseSpace LEDs on a pair of stereo glasses, which is worn by the participant during our experiment (see Figure 6.5). We carefully

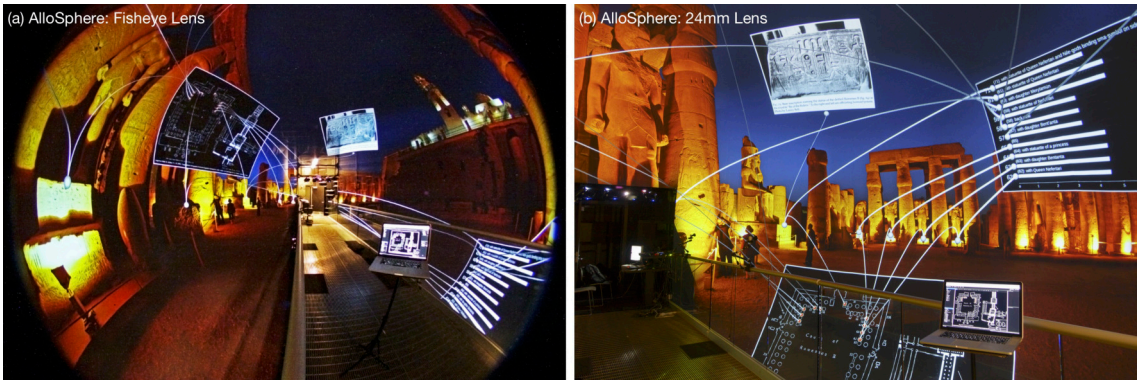


Figure 6.6: Annotations on a stereo panorama taken by Tom DeFanti *et al.* [195] at the Courtyard of Ramses II, part of the temple of Luxor in Egypt. (a) Full-surround version in the AlloSphere, captured using a fisheye lens. (b) Captured using a 24mm lens.

placed the LEDs such as not to interfere with the user's maximum field of view. From the head tracking information, we can easily derive which part of the annotations is to be shown to the user and which part is to be hidden given a desired field of view.

We display a black rectangle frame at the border of the field of view to make the user aware about this limitation. Figure 6.5 shows an example of a limited field of view.

Simulating tracking artifacts Simulating tracking artifacts is a bit more involved than simulating field of view. Since our surround display is fixed, we have to simulate the latency and jitter of augmentations as if a display were worn by the user based on their head movement. Let the true head orientation be denoted by the quaternion $q(t)$; the head orientation estimated by the simulated AR device be denoted by the quaternion $q'(t)$, then the displacement of the augmented content shown on the screen is given by $q'(t)q(t)^*$, as illustrated in Figure 6.7[1].

When simulating an AR device with perfect tracking, we have $q(t) = q'(t)$, so there is no displacement of the augmentations on the full-surround display.

Some latency is introduced by the PhaseSpace tracking system. Therefore, the latency

[1]Here we use quaternions to represent rotations. Readers unfamiliar with quaternions can treat them as 3×3 rotation matrices, and treat $q(t)^*$ as matrix inversion.

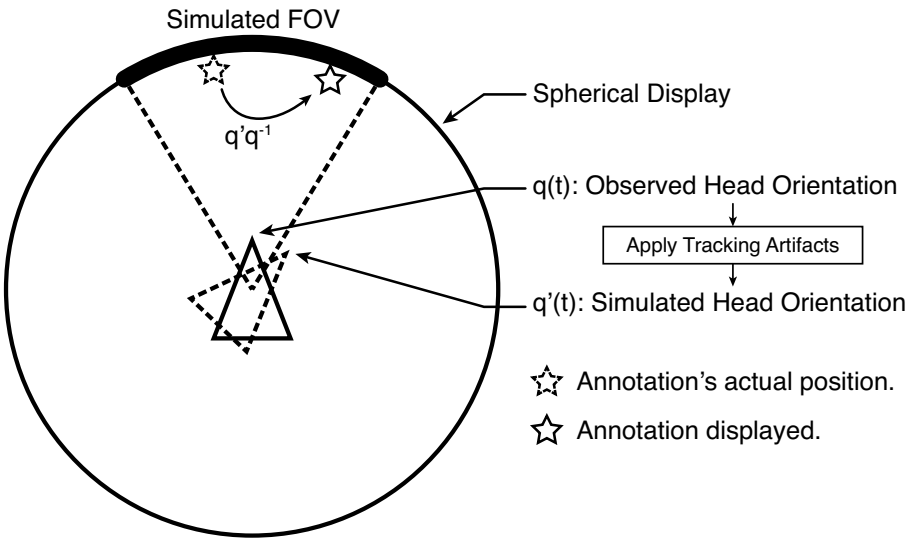


Figure 6.7: Simulating an augmented reality device with virtual reality in a full-surround display.

we simulate is delayed by the PhaseSpace system latency. However, when simulating perfect tracking, since there is no displacement of the augmented content at any time, the PhaseSpace latency has no effect on augmentations (only on the exact placement of the FOV frame in case of a limited FOV) and augmentations are rock-solid. In addition, previous work on the validity of AR simulation [197] suggests that simulator latency is an additive effect on artificial latency, thus we can consider them together. We measured the end-to-end latency of the system (the sum of tracking system latency, network latency and rendering latency) using a high-speed camera at 100fps. The result is 80 ± 20 ms.

6.6.2 Limitations

While the PhaseSpace system tracks 6 degree of freedom head pose, for our study we decided to assume a single viewing position in the center of our spherical display. This is because the stereo panorama that acts as the real world cannot respond to position changes. Hence, the scene will be distorted if users move away from the center. However, the surround display, with a 10m diameter has a fairly big sweet spot, so distortions

are very limited on small movements. This setup is however not suitable for tasks that require walking. In our experiment, we placed all the annotations around the participant, and instructed the participant to stay close to the center of the sphere.

Likewise, with this setup, it is hard to display objects that are very close to the participant. This is because we cannot guarantee that the user's head is at the exact center of the sphere, and position inaccuracies are more pronounced when the object is very close to the user. Hence, in our experiment, we avoided annotations that are too close to the participant.

The FULL FOV condition we simulated is partially occluded by the frame of the 3D shutter glasses, which are measured to have an approximately 108×82 degree FOV. Users can still see part of the augmented scene in full human FOV underneath and beyond the frame, but the view is more limited than by most vision-correcting eyewear.

The simulation of the real world is currently presented as a still stereo panorama. No motion effects were employed for real-world simulation in this study.

6.7 Implementation Details

The software architecture of our immersive visualization system is shown in Figure 6.8. iVisDesigner consists of a web-based interface for visualization construction and a web server in Twisted and Django. In our immersive version, the web server also acts as a gateway to the AlloSphere infrastructure. Rendering in the AlloSphere is performed on 13 rendering machines, each of which runs a Scene Renderer that renders the 3D scene using a cubemap approach. The visualization renderers are launched by the scene renderers, each of which renders a single visualization.

The renderers in the AlloSphere are written in Node.js with custom bindings to the AlloSystem libraries and the Skia graphics library. This allows us to reuse the iVisDesigner

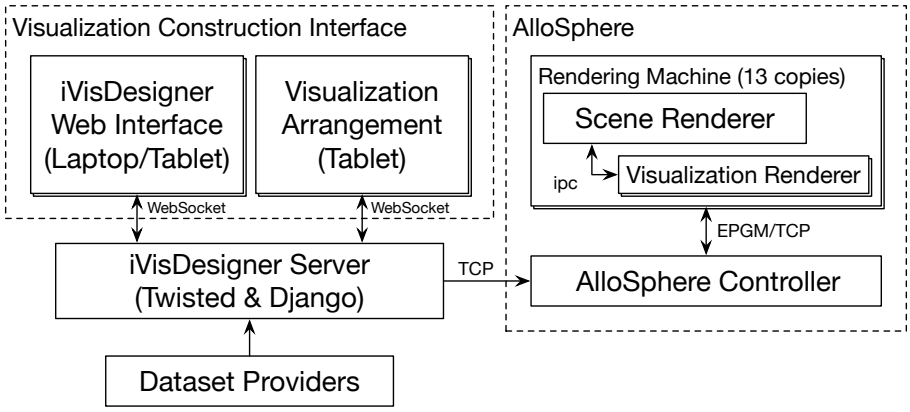


Figure 6.8: The architecture of our content creation system.

code that works in web-browsers to render visualizations in the AlloSphere.

The laptop and tablet interfaces are programmed with HTML5 technologies (HTML, CSS, JavaScript); the web server and dataset providers are written in Python with Twisted and Django. For the AlloSphere part, we used Node.js with custom modules and OpenGL bindings, which allows us to reuse iVisDesigner's rendering and synchronization code from the web-based interfaces for rendering visualizations in the AlloSphere infrastructure. There are 13 rendering machines currently used in the AlloSphere, each connected to two projectors. Node.js uses a single-thread event-driven model, which means each process can only run one thread. To better utilize the multiple processors available in the rendering machines, we used a multi-processes model for rendering. For each rendering machine, there are a set of visualization rendering processes, and one "Scene Renderer" process that collects the output from the visualization rendering processes via IPC shared memory and renders them in the 3D space (one cubemap for each eye), and then performs necessary wrapping and blending to render the scene correctly to the projectors. The controller in the AlloSphere servers communicates with the iVisDesigner server, receiving dataset and visualization updates and then broadcasting them to the rendering machines.

Additionally, we created a testbed program for normal desktop computers (see Figure 6.6 (a, b). This testbed allows us to view the visualizations for debugging purposes or demonstrate our system to others without being inside the AlloSphere. It can render the 3D scene in either perspective or equirectangular projection mode, while supporting monocular, anaglyph, and dual viewport stereo modes. We also added a bit of live-coding functionality to the system, allowing users to write code in the web-based interface and send it to the rendering machines. Although this is mainly for testing and debugging purposes, it does allow advanced users to experiment with their ideas quickly.

Chapter 7

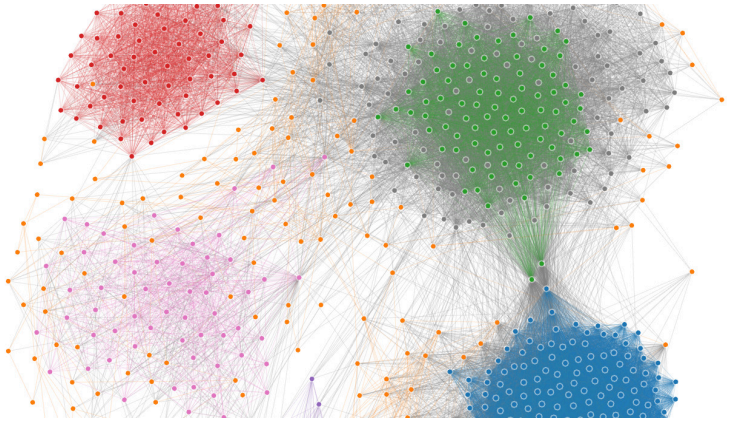
Stardust: A Library for Cross-Platform GPU-based Visualization

7.1 Introduction

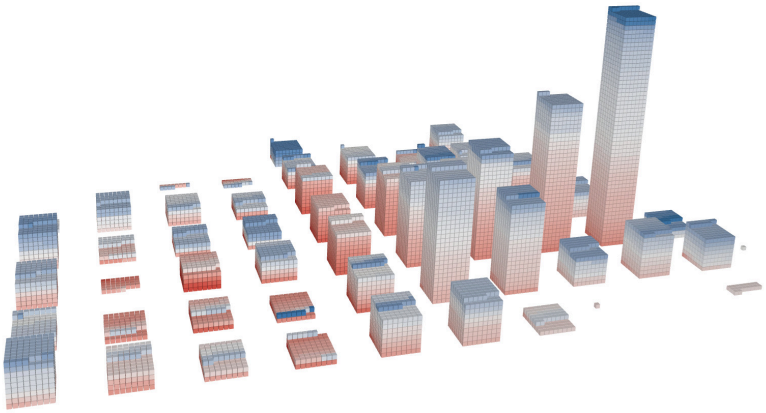
Immersive analytics has recently begun to gain attention in the field of information visualization [198, 199, 200]. Compared to desktop or laptop screens, immersive environments allow many more visual elements to be rendered because of the vast virtual spaces they provide. Furthermore, they come in many different form factors such as HMDs, large displays [127], CAVEs [128], and curved/spherical displays. Therefore, not only high performance rendering but also support for diverse display systems are desired in future visualization libraries. Supporting diverse display environments requires a more flexible way of projection and distortion, as well as a uniform representation of both 2D and 3D content.

There have been continuous research efforts on visualization libraries and frame-

The contents of this chapter have been previously published in *Computer Graphics Forum*.



(a) Node-link diagram of Facebook users, 747 nodes and 60,050 edges



(b) Instance-based visualization with 3D stacking, 31,618 items

Figure 7.1: Example visualizations created using the Stardust library for GPU-based rendering. For more examples, see Section 7.4 and visit https://stardustjs.github.io/examples.

works to facilitate the creation of information visualizations (*e.g.*, [27, 31, 6, 7, 36, 37]). Thanks to the recent advancement in web technologies (*e.g.*, HTML5, SVG, and D3), and modern browsers that support them, it has become very common to create visualizations for the web environment. Specifically, since its introduction in 2011, D3 has been widely utilized by visualization creators targeting the web. In addition, several higher-level visualization libraries and frameworks (*e.g.*, C3.js [201], and NVD3 [202]) have been developed using D3 for rendering graphical marks.

However, due to its dependency on the DOM tree and SVG, D3 cannot effectively

handle a large number of graphical marks, especially when animating them. For example, in a test on rendering animated scatterplots, D3 can deal with up to about 2,000 points in real-time (faster than 24 frames per second) on a modern personal computer [203]. Vega [36] implements its own rendering backends (Canvas or SVG) and performs optimizations on the dataflow model. However, due to the necessity of manipulating or rendering individual marks, it is still hard to achieve high performance on a large number of marks. Visualization libraries based on imperative rendering (*e.g.*, Processing [26] and p5.js [26]) have a similar performance drawback. While utilizing GPUs can drastically improve rendering performance [204, 41], GPU programming has a steep learning curve and requires a considerable knowledge of the computer graphics pipeline and shader programming. Hence, visualizations that require handling a large number of graphical marks are currently inaccessible to visualization creators who do not have expertise in computer graphics, and thus not yet fully explored by the community.

To address these issues, we designed and developed *Stardust*, a new library that leverages GPU processing power for information visualization rendering while providing a familiar and transparent programming interface similar to D3. Stardust is designed to support both 2D and 3D visualizations on various display environments through the same programming interface. Stardust consists of a core library and a set of platform libraries. The core library contains a set of predefined visual marks and provides a TypeScript-like programming language for developers to specify custom ones. Mark specifications are compiled to an internal representation that describes input attributes and rendering process. The core library also provides a data-binding API and a set of scales similar to D3's. The platform libraries take the compiled internal representations and the developer-specified data bindings, apply display-environment-specific view transformations, compile them into appropriate GPU shader code (*e.g.*, GLSL, HLSL) and buffers, and finally execute the rendering commands.

```
(1) let platform = Stardust.platform("webgl-2d", canvasElement, width, height);

(2) let polyline = Stardust.mark.create(Stardust.mark.polyline(), platform);

(3) let positionScale = Stardust.scale.custom(`
      Vector2(
        (R - r) * cos(value) + d * cos((R / r - 1) * value),
        (R - r) * sin(value) - d * sin((R / r - 1) * value)
      ) * 50 + Vector2(250, 250)
    `);
    positionScale.attr("d", 2.19).attr("R", 5).attr("r", 5 * (18 / 41));

(4) polyline.attr("p", positionScale(d => d * 41))
            .attr("width", 2).attr("color", [ 0, 0, 0, 0.3 ]);

(5) polyline.data(_makeRange(0, Math.PI * 2, 10000));

(6) polyline.render();
```

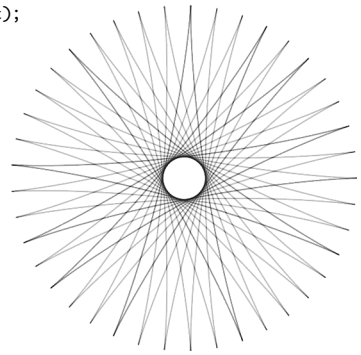


Figure 7.2: Plotting a parametric function (Hypotrochoid) with Stardust. Left (1) creating a platform object with the given "canvasElement" using the WebGL platform; (2) creating the polyline mark; (3) creating a custom scale for position; (4) setting the attributes for the polyline; (5) assigning data (here we create an array of 10,000 numbers from 0 to 2π); and (6) rendering the polyline. Right: The resulting plot. Refer to the supplemental material for an animated version of this example.

The goal of Stardust is not to replace existing visualization libraries and frameworks but to complement them for efficient rendering of graphical marks. Similar to D3.js, we do not intend to provide a new visualization grammar (cf. The Grammar of Graphics [33] and Vega-Lite [37]), but to focus on creating a set of building blocks that is easy-to-use, transparent in terms of representation, and portable to multiple platforms.

To evaluate Stardust, we compare it against existing visualization libraries in terms of rendering performance on a modern GPU-equipped PC. Our results show that Stardust achieves 10–100 times speed boost compared to existing libraries when rendering more than 100k graphical marks. We also demonstrate five example visualizations to show its expressiveness (Figure 7.1, Figure 7.5, Figure 7.4) and the support of multiple platforms to show its portability. We develop Stardust as an open-source project (`https://stardustjs.github.io`) in the hope to further evaluate and improve its usability through adoption and feedback from real users in the future.

7.2 Stardust Design

We see Stardust as a complement to D3 instead of a replacement. Stardust is good at rendering a large number of marks and animate them with parameters, while D3 has better support for fine-grained control and styling on a small number of items. For example, to create a scatterplot with a large number of items, we can use D3 to render its axes and handle interactions such as range selections, and use Stardust to render and animate the points.

7.2.1 Design Rationales

DR1. Representational Transparency and Familiar API A transparent representation makes it easy for developers to reason about outcomes (*i.e.*, the resulting visualization), and thus reduces programming and debugging efforts. For example, D3 removes the intermediate layer between user programs and the underlying API by directly binding data to DOM elements. In Stardust, we strive to create an API with similar representational transparency. In addition, we designed the Stardust API to be as close to D3's as possible, while utilizing the GPU for rendering. We believe that the similar API makes it easier for developers to adopt Stardust by transferring their existing knowledge on D3.

DR2. Declarative Data-Binding over Imperative Evaluation D3's imperative evaluation model makes it easier to debug because there are no hidden layers of data flow that complicate the developers' mental model. In GPU programming, however, data binding and shader uniforms have to be fully prepared before one can render anything. This means one cannot render any graphical marks before all attribute bindings are specified. In addition, there is no web inspector for GPU-rendered graphics to provide a similar debugging experience as with D3. Therefore, we decided to use declarative data-binding

instead of imperative evaluation.

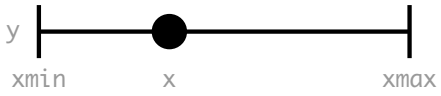
DR3. Being Platform Agnostic One important design goal of Stardust is to be able to target multiple display environments. Thus, we designed Stardust to be agnostic with respect to platform details including (1) the graphics API (*e.g.*, OpenGL in Unix-based systems and Direct3D in Microsoft Windows) and (2) the rendering and projection method. For example, there are multiple stereoscopic rendering techniques, including tilted view frustums, Omnistereo [135]. For 3D contents, there are also different shading techniques and post-processing.

7.2.2 API Design

Graphical elements are called "marks" in the Stardust API. The units of operation in Stardust are arrays of marks (*e.g.*, Figure 7.2-2). Each array of marks has (1) a specification of the mark type; (2) attribute bindings that declare how data items map to marks' input attributes (*e.g.*, Figure 7.2-4); and (3) an array of data items (*e.g.*, Figure 7.2-5). Stardust uses a declarative model with lazy evaluation that is different from D3's selection-driven approach where operations are imperatively evaluated (DR2). In Stardust, actual data bindings happen at the `render` call (*e.g.*, Figure 7.2-6), where Stardust creates GPU shaders and uploads data to GPU buffers for rendering. To be independent of platform details (DR3), Stardust does not rely on any underlying scene graph representation such as the DOM model operated by D3, avoiding the complexity of D3's `enter` and `exit` operations. Updates to the array of data items can be done with array operations.

Mark Specification To make it easy to create basic visualizations, Stardust provides a set of predefined marks, including `circle`, `rectangle`, `line`, `polyline`, and `wedge`. For example, in Figure 7.2-2, we use the `polyline` mark to plot the parametric function.

Because the expressiveness of predefined marks is limited, we provide a mark specification language based on TypeScript [180], which is a typed superset of JavaScript (DR1). Custom marks can be created by writing a function with input attributes, intermediate computation steps, and emit statements. For example, the code below creates a custom mark — a range bar with a circle:




```
// Import predefined marks
import { Circle, Line } from P2D;
// Define a utility mark
mark VLine(x: float, y: float) {
  Line(Vector2(x, y - 3), Vector2(x, y + 3), 1);
}
mark RangeBarWithCircle(
  x: float, y: float,
  xmin: float, xmax: float
) {
  // Emit the VLine mark defined above
  VLine(xmin, y);
  VLine(xmax, y);
  // Emit predefined marks, Line and Circle
  Line(Vector2(xmin, y), Vector2(xmax, y), 1);
  Circle(Vector2(x, y), 2);
}
```

The mark specification language exposes the details of marks (DR1). For the marks composed of other marks, developers can follow the function calls until they reach the specification of predefined marks, which are also written in the same language.

Stardust currently supports variables, basic expressions, if-else statements, fixed range for-loops, and function calls. It enables developers to represent a wide range of useful marks including common shapes, parametrized glyphs, Bézier curves, ribbons, wedges, and polylines. More language constructs such as arrays, lambda functions, and classes are left for future implementation work.

Attribute Binding Similar to D3's data-driven selections, Stardust allows mapping data items to marks with user-defined attribute bindings. Attribute bindings can be specified in the same way as D3's `attr` operator (DR1); all D3 scales that return numerical values (*e.g.*, `linear`, `log`, and `time`) as well as layout modules including `force` and `treemap` in D3 can be used in Stardust.

In addition to using D3's scales, Stardust introduces *compilable* scales (DR2). Stardust provides predefined scales including linear, logarithmic, and color interpolation, which have parameters such as `domain` and `range` similar to their counterpart in D3. Besides predefined scales, Stardust allows custom scale expressions and attributes. For example, in Figure 7.2-3, we declare the positions of the polyline points using the parametric function.

In contrast to D3's immediate scale evaluation at the time of data binding, Stardust's compilable scales are transformed to shader code to be executed in GPU. This allows for adjusting parameters (such as the domain and range of a scale) without having to go through all the data items again and upload updated data to the GPU. Since there is currently no support for introspection in JavaScript, Stardust's scale API is different from D3's. For example, in Figure 7.2-4, the lambda function is inside the Stardust scales, which is different from D3's pattern.

Stardust introduces `Array`-typed attributes, which are implemented as textures in the GPU. Developers can use these objects to store additional data attributes or parameters and reference them in the code. This is very useful when the amount of changing data is small but the number of graphical marks to render is large. For example, in a node-link visualization of a relatively dense graph (*e.g.*, a social network), there are $O(n)$ layout parameters, but usually $O(n^2)$ edges to render. Storing the layout parameters using `Array` objects and referencing them while rendering the edges results in a much better animation performance.

Rendering, Animation, and Interaction After specifying all required attributes and data items, simply calling the mark's `render()` function draws the items. The `platform` object (*e.g.*, Figure 7.2-1) manages the platform-dependent details such as camera position and projection matrices (DR3).

Attributes that are defined as constant values (including `Array` attributes) can be changed after the `render` call without having to rebuild the GPU shader nor uploading data to the GPU. This helps creating parametric animations such as transition and morphing. For example, the parametric plot in Figure 7.2 can be animated by changing the `positionScale`'s `d`, `R`, and `r` attributes.

D3 uses events directly from DOM elements, such as `mousedown` and `mouseup`, to support interactions. These events occur on the item level where programmers can easily figure out corresponding marks and data items. However, they are not available in GPU-based graphics APIs including WebGL. Stardust supports item-picking by introducing another pass that renders marks into an item-picking render buffer. We assign a unique index to each mark based on the rendering order, and in a buffer, we use the four 8-bit color channels to encode the index of the rendered mark at each pixel; we can encode up to 2^{32} marks. Then, we retrieve the corresponding data item by decoding the pixel's color to convert it back to the index.

7.3 Architecture and Implementation

We have designed Stardust using a modular structure and a platform-independent internal representation (DR3) that acts as an intermediate layer between the developer-facing API and display-environment-specific internal code.

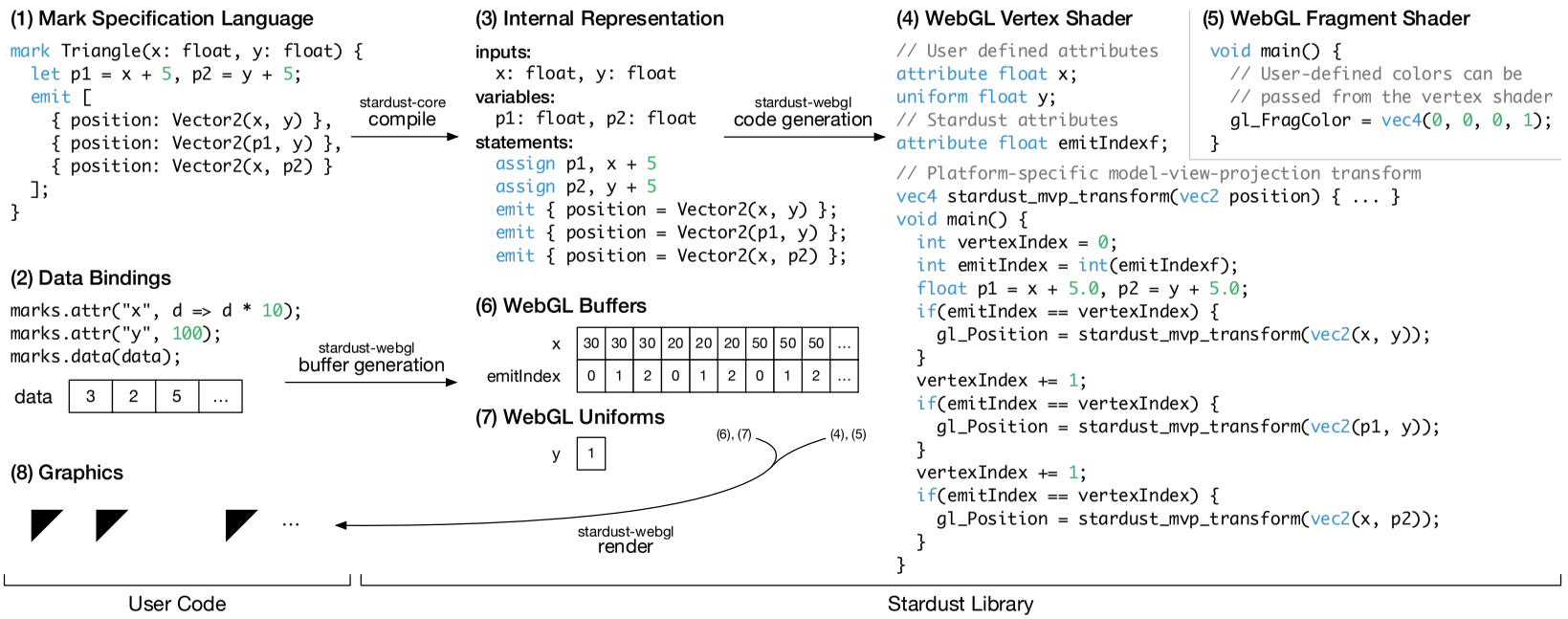


Figure 7.3: Stardust's rendering process: User code creates mark specifications and data bindings (1, 2). The mark specification is compiled into an internal representation (3). The platform backend, in this case stardust-webgl, converts the internal representation to WebGL shaders (4, 5), and converts the data bindings to WebGL buffers and uniforms (6, 7). Stardust executes the WebGL shaders along with its buffers and uniforms in the GPU to produce the graphics (8).

Modular Structure Stardust consists of the `stardust-core` module and a set of platform modules. The core module defines the developer-facing API, built-in marks, and scales, and contains a compiler that converts the mark specification language to Stardust's internal representation. The platform libraries transform internal representations into shader programs, and manage shader uniforms and GPU buffers for rendering. The modular structure allows developers to choose only the parts they need (*i.e.*, core plus desired platforms), and thus minimize the library's download size. Developers could also write support modules for additional mark types, scales, and helper functions. As a demonstration, we have created the `stardust-isotype` module, which supports importing SVG files as isotype [205] marks.

Compiling to an Internal Representation The compiler in the `stardust-core` module transforms mark specifications (Figure 7.3-1) to Stardust's internal representations (Figure 7.3-3), which are encoded as JSON objects, and thus can be easily serialized and deserialized (DR3). A mark's internal representation consists of (1) a set of input attributes, which can be bound to data using Stardust's data binding API; (2) a set of internal variables, which store computation states; (3) a set of output attributes; and (4) a list of statements. The statements can be assign statements, conditional statements, loop statements, or emit statements. Conceptually, a mark is rendered by setting the input attributes according to user-specified data bindings, executing the statements, and finally capturing the emitted vertices. Each triplet of consecutive emitted vertices forms a triangle, which is rendered to the screen.

The compiler parses the input code and transforms the resulting abstract syntax tree (AST) to the internal representation by inlining function calls and (optionally) unrolling loops. The compiler also performs a necessary type inference and inserts code for type conversions. When a custom scale is used for a mark attribute, the compiler inserts the

scale code and scale attributes to the internal representation, and then assigns the result of the scale to a variable for the mark attribute. In this way, the compiler produces a single piece of internal representation that can be passed to the code and buffer generation step as described below.

Code and Buffer Generation Stardust's internal representation serves as a platform-independent representation of marks, from which Stardust's platform modules (*e.g.*, `stardust-webgl`) perform the final code generation step to produce GPU shader programs (Figure 7.3-4,5). Stardust first determines the forms of input attributes. It converts the mark attributes that are specified as immediate values (*e.g.*, `.attr("y", 100)`) to GPU shader uniforms (Figure 7.3-7), and the mark attributes that are data-driven (*e.g.*, `.attr("x", d => d * 10)`) to vertex attributes along with the corresponding GPU buffer (Figure 7.3-6). Stardust employs lazy update to prevent sending the same uniforms and buffers multiple times. Stardust re-writes the statements in the internal representation using GPU's shader language (GLSL in the case of WebGL), and executes the shaders and associated uniforms and buffers in the GPU (*e.g.*, glDrawArrays) to render the final geometry.

The platform implementation adds its own geometry transformations. For example, in Figure 7.3-4, the `stardust_mvp_transform` function is inserted and called to produce the `gl_Position` values. Note that the geometry transformations are not necessarily matrix multiplications. For example, to render content in the AlloSphere, we use its own Omnistereo [135] per-vertex displacement to achieve stereoscopic effects in all directions. The emit statements can be directly mapped to geometry shader's EmitVertex statements on platforms with geometry shader support. In platforms without geometry shaders (*e.g.*, WebGL and OpenGL ES), Stardust can convert the data format and re-write the shader program as described below.

Handling Emits in the Absence of Geometry Shaders Stardust's internal representation naturally maps to the geometry shader in the computer graphics pipeline. However, there are multiple versions of graphics APIs and shading languages, among which only advanced versions support geometry shaders. For example, WebGL has no support for geometry shaders in its current version. Thus, Stardust's platform implementation converts the geometry generation process into a non-generative process when the target platform does not support geometry shaders. Stardust accomplishes this by expanding the mark generation process through program transformation. Originally each mark corresponds to a single data item from which input attributes are derived (*e.g.*, in Figure 7.3-2, there are three data items, each corresponds to a mark, and the lambda function `d => d * 10` derives a mark's `x` input attribute from the data item). Stardust duplicates the input attributes such that each generated vertex becomes one standalone vertex in the vertex array, increments a vertex emit index on each repetition to differentiate the vertices (*e.g.*, in Figure 7.3-6, each data item is transformed by the lambda function and then repeated three times, with an added `emitIndex` attribute), and finally re-writes the vertex generation process to use the emit index to determine which vertex to display. Figure 7.3-4 illustrates how standalone emit statements are transformed, and the code below shows how loops are transformed:

The original for loop:

```
// (input attributes)
for(let i in 2..7) {
    emit F(i);
}
```

Transformed code:

```
// (input attributes)
// emitIndex attribute
// vertexIndex, i variable
vertexIndex = 0;
// For loops without state
// update can be simplified
// to run only one step:
i = emitIndex - vertexIndex + 2;
if(i >= 2 && i <= 7) {
    emit F(i);
}
```

Note that this conversion is performed at the internal representation level so that similar platforms can share it.

7.4 Examples

With Stardust's predefined marks (circles, lines, areas, and wedges), we can create simple visualizations such as 2D/3D bar charts, scatterplots, line charts, pie charts, area charts, and parallel coordinates. In this section, we show a diverse set of more complex visualization examples to illustrate Stardust's expressiveness, and discuss a coding playground we have created to demonstrate Stardust's portability.

7.4.1 Instance-based Visualization

Researchers have explored instance-based visualizations or unit visualizations, which explicitly represent each data item from the dataset (*e.g.*, [206, 207, 205, 44]). Here, we illustrate the power of Stardust by animating and interacting with instance-based visualizations.

First, we show a reproduction of animated transitions similar to SandDance [44]. The visualization shows voting results in the United States' 2012 election; among 17 attributes we use longitude, latitude, state, and percentage of Obama supporters. In the visualization, we draw the data items as cubes with three different layouts including (1) a scatterplot of longitude and latitude, (2) an instance-based bar chart binned by states, and (3) a 3D stacked bar chart binned by both longitude and latitude (Figure 7.1b). The cubes are colored by the percentage of Obama supporters. To implement the visualization, we first wrote JavaScript code to compute the bin and within-bin index of each data item for layouts (1) and (3). Then, we wrote marks in the mark specification language. We wrote three functions to compute the positions of cubes based on the three layouts, and

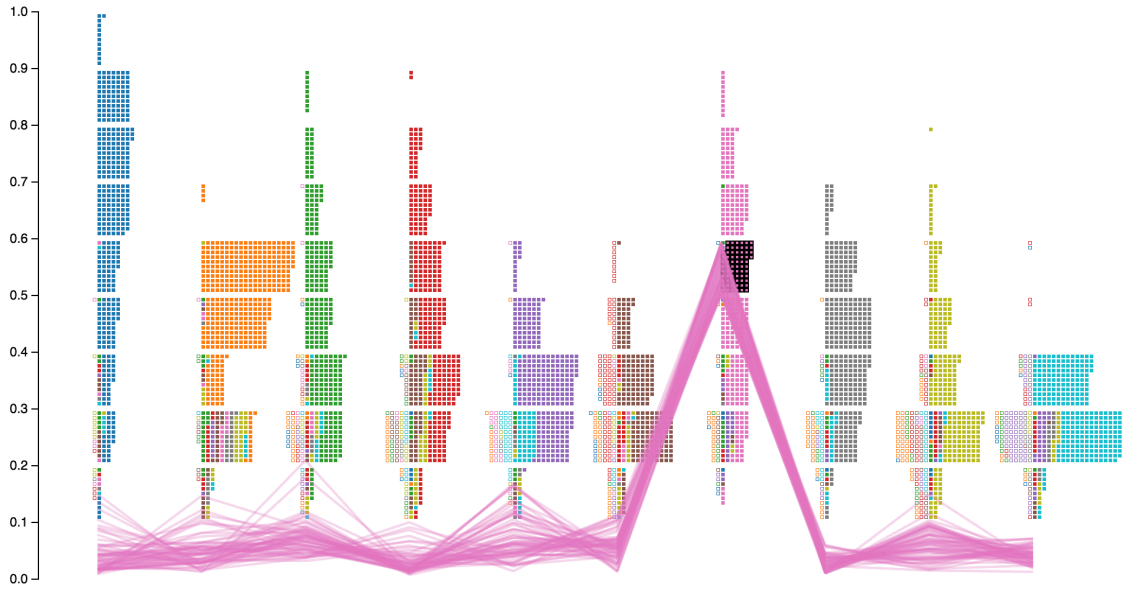
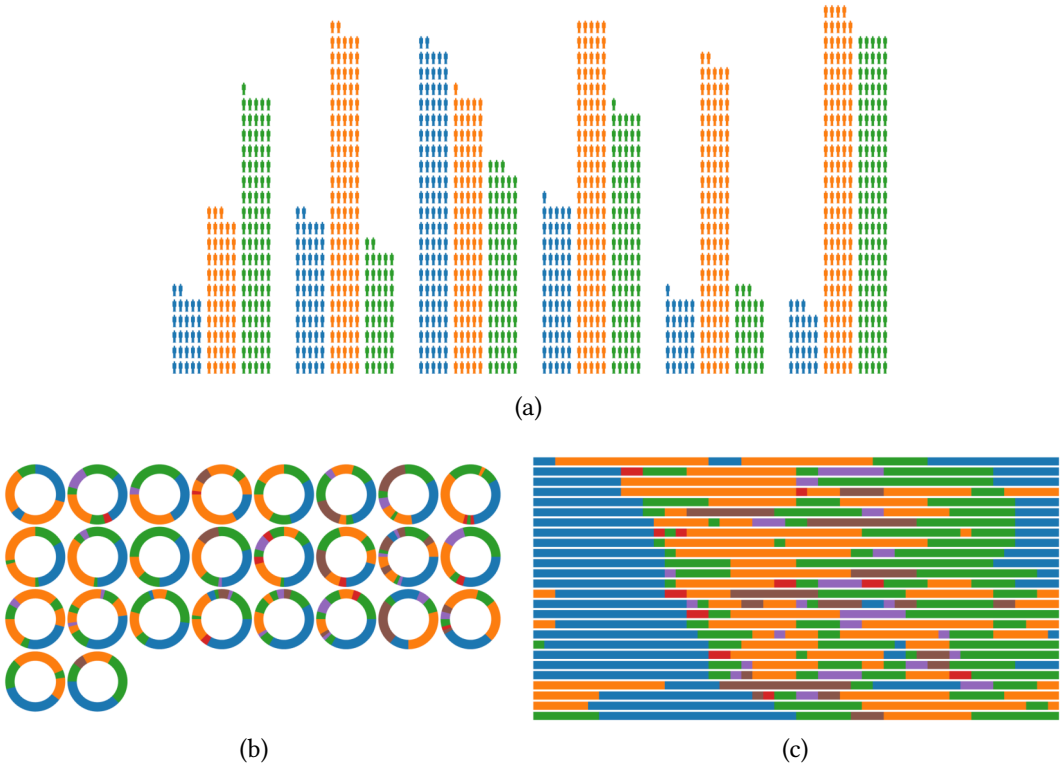


Figure 7.4: Reproduction of the Squares visualization [208] using Stardust.

in the main function we interpolate between these positions based on the animation time parameter. This example visualization has 31,618 cubes, and the transitions can be rendered at the monitor's refresh rate (60fps) on the computer we used for our performance evaluation (see Section 7.5).

Instance-based visualizations become more complex when we apply multiple levels of grouping, combine them with other visualizations, and add interaction. We use Stardust to render the Squares [208] visualization designed for multiclass classifier performance analysis (Figure 7.4). The Squares visualization design encodes instances as square marks, bins the squares according to their prediction scores, groups them by the predicted classes, and colors them by the labeled classes. Upon selection, parallel coordinates are shown to reveal detailed prediction scores. We use Stardust's item-picking functionality to implement the selection interaction. The version created with Stardust allows parameter changing and interactions at a much more responsive rate than the original D3-based version.



(a) (b) (c)

Figure 7.5: Reproduction of Isotype visualization (a) and *"The Daily Routines of Famous Creative People"* in circular layout (b) and linear layout (c) with Stardust.

7.4.2 Isotype Visualization

Stardust's custom mark specification allows developers to import marks from other sources. As an example, we have implemented the `stardust-isotype` module, which can load SVG files and convert them to mark specifications. The converted isotype marks have attributes including `position`, `size`, and `color`, which can be bound to data. Stardust thus supports isotype visualizations [205]. Developers can design isotypes in vector graphics tools and then export them as SVG files. Figure 7.5a shows our example isotype-based multi-column bar chart. It can perform instance-based animation between two versions (see the supplemental video for the animation).

7.4.3 Morphing between Circular and Linear Timelines

To show Stardust's representational power, we reproduced the visualization of *The Daily Routines of Famous Creative People* [209] with Stardust (Figure 7.5b, Figure 7.5c), and added a morphing animation between the linear layout and a circular layout as in [71] (see the supplemental video for the animated transitions). We implemented the marks with Stardust's `Wedge` object. Wedges can also express the linear layout because rectangles are wedges without any curvature. We implemented the morphing animation by linearly interpolating the wedges' parameters between the circular layout and the linear layout. We also used the item index to modulate the animation effect such that animation for different circles/lines happen at different times. The animated transitions between the two layouts can be easily rendered at 60fps on the computer we used for our performance evaluation, even with synthetic data of 100 times more wedges to render ($100 \times 237 = 23,700$ items).

7.4.4 Real-time Force-directed Graph Visualization

In graph visualizations, algorithms have been developed to accelerate the force-directed layout process [210]. For example, in D3's implementation, a quadtree structure is used to accelerate charge interaction using the Barnes-Hut approximation [211]. However, after layout algorithms have been accelerated, rendering becomes the bottleneck for animating graphs. In this example, we use Stardust instead of D3 to render a social-network graph of 747 nodes and 60,050 edges (Figure 7.1a). Each time the graph layout is updated by the algorithm, we re-assign the data to the nodes and edges, and schedule a re-render of the visualization. The layout algorithm alone can run at 145fps. The Stardust version achieved a significantly better frame rate of 60fps than D3's 1.6fps. Furthermore, this example achieved a frame rate of 21fps on a Google Pixel smartphone with Android

7.1.1 running Google Chrome browser version 56.0.2924.87.

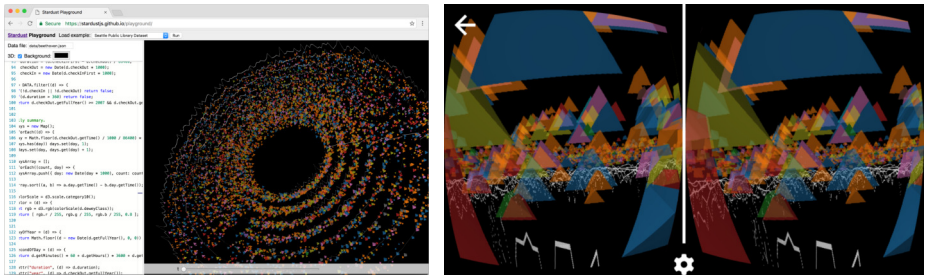
7.4.5 Coding Playground for Multiple Display Environments

We have created a coding playground, which enables developers to write code in a web-browser and run it *unmodified* in four different display environments; 1) the browser itself, 2) mobile VR with Cardboard viewers, 3) HTC Vive, and 4) the AlloSphere [136], a full-surround immersive VR environment (Figure 7.6). This playground is a starting point for building a cross-platform visualization authoring environment. For example, in the future we can explore how to easily design visualizations that can adapt to different levels of processing powers by using aggregation or subsampling techniques: we can show an overview visualization in mobile devices, while showing individual data items and allowing multiple coordinated views in immersive environments. Stardust provides a simple visualization representation for various levels of graphics hardware and rendering process behind different display environments.

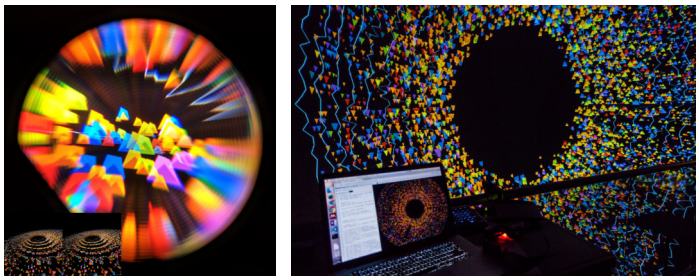
We implemented the coding playground's user interface using TypeScript and React. It embeds the Monaco text editor for syntax-highlighting and intellisense features. The visualizations in the web browser are rendered in `iframe` elements to minimize the interference between developer code and playground code. In the AlloSphere, we created a web server to receive code and commands from the user interface and coordinate 13 rendering machines that power 26 projectors.

7.5 Performance Evaluation

We evaluate Stardust's performance in terms of the initialization and rendering times on three types of visualizations.



(a) Browser (b) Cardboard



(c) HTC Vive (d) AlloSphere

Figure 7.6: A sample visualization created with Stardust in the coding playground. Developers can create 2D/3D visualizations for three different display environments with the same code. (a, b) run in browsers with WebGL; (c, d) run in the AlloSphere's Allofw framework with OpenGL 3.3; and (d) uses Omnistereo for a 360 degree stereo effect.

7.5.1 Conditions

We compare Stardust against D3, Vega (using Canvas backend), and the standard HTML5 Canvas. D3 is one of the most commonly used libraries in web-based visualizations. Vega is a very promising declarative visualization grammar with a performance-optimized reactive dataflow architecture [36, 37]. We choose Vega's Canvas backend because it is measured to be more efficient than the SVG backend. The HTML5 Canvas is the basis of many visualization libraries such as Processing that are not SVG-based. We implement three visualizations that use common mark types in information visualization: (1) scatterplots with points, glyphs, and isotypes; (2) parallel coordinates with lines and polylines; and (3) node-link graphs with points and lines. The visualizations we created with D3 adhere to common D3 programming patterns, and the ones with Canvas are

written to access Canvas' API as directly as possible to eliminate any performance overhead introduced by extra representational layers. For example, to compute an `x` position, we choose `x = a * data + b` over `x = scale(data)` because `scale(data)` introduces an extra function call.

To simulate animation and layout updates, we animate these visualizations by randomly changing scales for the scatterplots and parallel coordinates and shifting node positions for the graphs. We run these visualizations with sizes ranging from 100 to 1M (the size of a graph is measured as the number of edges, which dominates the rendered marks). The resolution of the visualizations is set to $1,000 \times 1,000$. We run our performance measurements with an iMac Retina 5K 27-inch Late 2014 model with the macOS Sierra 10.12.1. The machine has a 4GHz Intel Core i7 processor with 32GB 1600MHz DDR3 memory, and an AMD Radeon R9 M290X graphics card with 2GB memory. The benchmark is performed on the Google Chrome browser version 55.0.2883.75 (64-bit).

7.5.2 Results

It is important to differentiate two types of rendering times: (1) initializing and rendering the first frame, which directly influences the page load time; and (2) rendering subsequent frames in response to parameter updates, which is crucial for parameter-based animation. Because modern web browsers commonly use background threads for rendering, especially for WebGL-based rendering, it is not possible to measure frame time directly by timing the API calls. We used the `requestAnimationFrame` API to measure the time duration between adjacent frames and among multiple frames for initialization and rendering, respectively. This approach, however, has a minimum measurable unit, which is the refresh rate of the monitor because browsers wait for the next monitor refresh before performing an "animation frame" when rendering is faster than the re-

fresh rate. On the machine we tested with, this minimum unit is 16.7ms. Therefore, all values we report here are more than 16.7ms whose corresponding frame rate is 60fps. To model parameter updates during rendering, the rendering time is measured by averaging 30 frames with random visual and layout parameters. We also run each trail 10 times and take the average to minimize performance fluctuations. To avoid potential garbage-collection delays, we restart the browser for every trail larger than 10,000 items.

Our performance simulation results show that Stardust is faster in both initialization and rendering for large numbers of marks (Figure 7.7). The time for initialization and rendering the first frame is around 2x faster than Canvas, and for subsequent frames Stardust is 10–100x faster than Canvas, Vega, and D3. However, Stardust's initialization time is slower than Canvas, D3, and Vega for small numbers of marks because Stardust has an overhead of shader compilation. In addition, we observed that while canvas is faster than D3 for the scatterplot and graph, it is slower than D3 for rendering the parallel coordinates. The fact that D3's render time for parallel coordinates looks quadratic and is slower than its initialization plus render time is notable. We suspect that this is caused by the Google Chrome browser's internal mechanisms for handling updates to the `d` attributes on SVG `path` elements.

7.6 Discussion and Future Work

Stardust is inspired by the observation of media artists creating 3D data visualizations that render and animate a large number of data-driven glyphs. This was very difficult in Processing; drawing the glyphs one by one was too slow even with WebGL/OpenGL, but putting them altogether into a mesh object made it impossible to parametrically animate the glyphs without using a vertex shader. We initially created a library that supports a set of predefined marks and D3-like data bindings. However, creating a new mark required

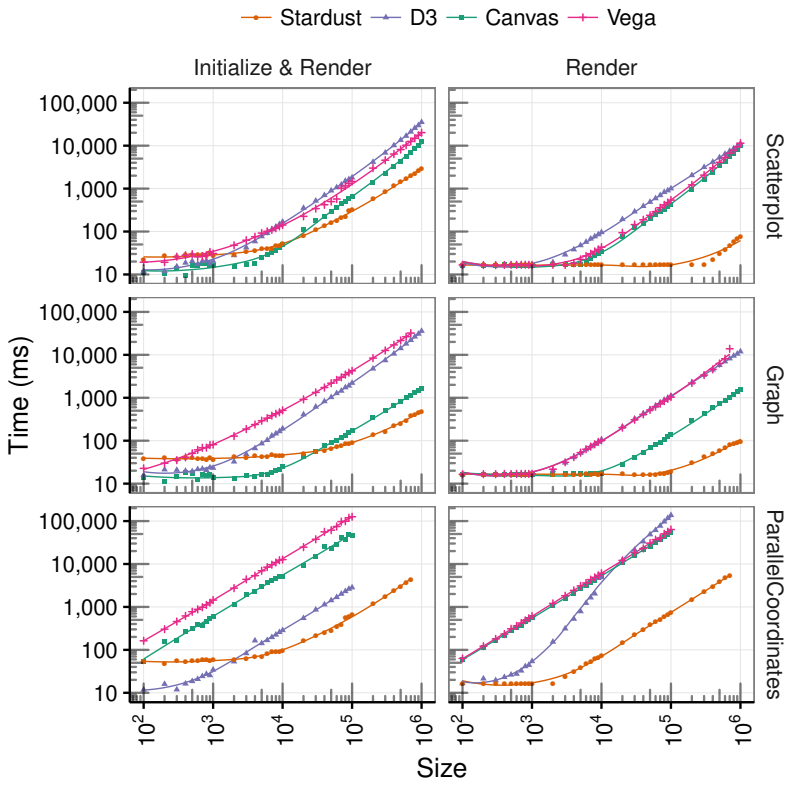


Figure 7.7: Performance evaluation results plotted in log-log scales.

a considerable knowledge of the library and shader programming, and geometry shaders used in the initial version is not currently available in web browsers. Stardust's internal representation and shader code generation architecture contribute to its expressiveness and portability. Together, they support a simple and uniform API while hiding complex platform-specific implementation details (*e.g.*, the geometry generation process).

In this section, we discuss Stardust's limitations which form interesting directions for future work.

Pixel-space Specifications Like D3, the Stardust API currently only specifies geometry with a few styling options including color, opacity, and shading options. Although these cover a large set of information visualizations, other visualizations require advanced styling of marks such as pattern-encoded marks and textured marks. Therefore, it would

be useful to extend Stardust to support pixel-space specifications.

Per-mark State To be platform-agnostic, Stardust only abstracts the data-driven geometry generation process without relying on scene-graph models, which naturally keep mark states in nodes. Because Stardust's geometry generation process on GPU is currently stateless, it cannot provide a support for some of the D3 functionalities such as filtered selections and enter/exit selections. Only data-bindings that are specified as immediate values can be changed without having to re-compile shaders and re-upload buffers to GPU. Therefore, it is hard to implement animations that require per-mark state recording such as particle system simulations in Stardust. Having an API support for per-mark state storage, update, and retrieval is important for future work.

GPU Support for Data Transformation and Aggregation Stardust currently supports only GPU-based rendering. Similar to D3, to use Stardust, data has to be aggregated so that each data item *independently* maps to a mark. For example, in our instance-based visualization example (Section 7.4.1), we have to compute the binning in JavaScript and assign a bin and a within-bin index for each data item. In the future, we would like to investigate how to support GPU-based data aggregation and transformation (*e.g.*, [45]).

Optimization As a first step, we primarily address the problem of simple and easy specification. As we have not yet implemented optimization techniques, Stardust currently relies on the shader compiler's optimizations. For future versions of Stardust, we plan to explore optimization techniques.

Further Evaluation Given Stardust's simplicity and similarity to D3's API, we anticipate it can be easily adopted by visualization creators. As a public deployment could lead to more meaningful evaluations, we constructed a website that provides online code

examples and documentation to help developers get started. We hope to get feedback from visualization creators and improve the library.

7.7 Conclusion

We have designed and implemented Stardust, a GPU-based library for information visualization, as an open-source project (https://stardustjs.github.io). It provides a simple programming interface while targeting both regular and novel display environments. Stardust enables the creation of a variety of GPU-powered visualizations as demonstrated by our examples. It achieves a significant performance boost compared to the standard HTML5 Canvas, D3, and Vega on a large number of graphical marks.

Chapter 8

Idyll-MR: Declarative Authoring of Immersive Data-driven Stories

8.1 Introduction

While Stardust makes it easier for authors to program immersive information visualizations, the library focuses on providing the fundamental building blocks. In this chapter, we return to the topic of data-driven storytelling, which is an important way to communicate insight from data. Traditionally, visual data stories use two-dimensional media, such as a desktop display or a web page. Immersive visual data stories [118] can leverage immersive environments such as virtual and augmented reality as a medium to achieve the following potential benefits. First, immersive environments can utilize a vast amount of viewing and interaction space that can easily be viewed and navigated naturally and thus allow the creator of the story (author) to make use of the audience's inherent navigational capabilities. Second, structures that are inherently 3D can be better visualized, inspected, and understood in immersive environments. Third, immersive environments may provide opportunities for collaborative consumption of a story. In

this scenario, multiple viewers can perceive and interact with the story at the same time, choosing their respective viewing angles for complementary analysis. This provides opportunities for discussion among the audience that can potentially make the storytelling more effective.

While there is clear potential benefit of doing data-driven storytelling in immersive environments, the space is currently under-explored. One reason is that there is a lack of effective authoring tools that would enable easy creation of such stories. As a result, people resort to existing frameworks such as game engines (*e.g.*, Unity 3D) to create them. These general frameworks provide no or very limited constructs to specifically support visual data stories, such as data-driven visualizations and linked views. Some recent work [116, 117] has begun to fill this gap, and our efforts are complementary to theirs, and directly focused on supporting immersive data storytelling. Current tools and languages that are designed to support visual data stories, such as Idyll [192] and Ellipsis [69] only target 2D desktop or web browsers. Furthermore, stories created with these systems only run on a single computer and no explicit support is provided for collaborative analysis.

In this chapter, we present Idyll-MR, a novel authoring system that supports easy creation of immersive visual data stories. Idyll-MR extends Idyll [192] in the following ways: (1) Idyll-MR provides story components that are specific to immersive environments, including data visualizations, interactive widgets, and annotations in 3D space. (2) Idyll-MR provides layout components for the author to easily arrange story components in the immersive space. (3) Idyll-MR establishes a shared environment in which multiple viewers with different types of devices (including desktop computers and VR/AR headsets) can simultaneously perceive and explore the story.

The contributions of this work are the following:

- We present a set of components that allows an author to efficiently specify immer-

sive data-driven stories.

- We contribute the software architecture design and implementation of Idyll-MR, which features a React-inspired layer that supports efficient scene synchronization from a server to rendering clients through incremental updates.
- We demonstrate the expressiveness of the system through three examples that cover multiple usage scenarios.

8.2 Design of Idyll-MR

When we evaluated our options for choosing a language or authoring environment for immersive data visualization components, we surveyed the space of 2D and 3D data storytelling systems, and were impressed by the flexibility and declarative support of Idyll [192]. Idyll is a successful declarative language for creating web-based visual data stories, providing variables, derived variables, and components to the Markdown syntax. Below is some example Idyll code that illustrates these basic constructs. We direct the reader to Conlen *et al.*'s paper [192] to learn more about Idyll.

```
# Markdown Heading

Define a variable:
[var name:"x" value:5 /]

Compute a value from the variable:
[derived name:"x_squared" value:`x * x` /]

Show the computed value:
[Display value:x_squared format:"d" /].

Add a slider for the variable:
[Range value:x min:0 max:10 /]
```

Idyll supports a rich set of *components* for creating web-based interactive documents. A component is a building block of interactive documents, specified using the bracket

syntax: `[ComponentClass attr:"value" /]`. Idyll provides many built-in components, including several types of layouts, input widgets, and presentation elements. Authors can import existing React [181] components into their projects, or even write their own components using React.

We consider Idyll an ideal starting point for building an authoring language for immersive visual data stories, because: (1) Idyll is extensible. It allows any set of components to be used. The language itself is not tied to any existing library or framework; (2) Idyll supports variables and derived values, which are useful for building interactions between components; (3) Idyll can reduce the amount of overall programming effort and need for custom code [192]. Therefore, Idyll-MR is built as an extension of the Idyll language that enables the creation of *immersive* visual data stories.

In this section, we describe the design goals, language constructs, components, and implementation details of Idyll-MR.

8.2.1 Design Goals

Supporting immersive visual data story constructs There are many potentially useful constructs for immersive visual data stories. When designing Idyll-MR, we explored the set of such constructs, and implemented a set of Idyll components (see Table 8.1 and the Mixed Reality Components section below) to make it easy to create stories that incorporate such constructs.

Supporting co-located authoring and consumption While a web page can be consumed simultaneously by multiple viewers in front of the same computer, immersive environments currently require that each viewer use a separate device. Without synchronization at the application level, it is unlikely that multiple authors or viewers can share and discuss their experience even when they are at the same place (co-located). In

Idyll-MR, we aim to support co-located authoring and consumption of immersive visual data stories.

8.2.2 Overview of Idyll-MR Concepts and Functionality

Following the convention of Idyll, immersive visual data stories are specified as `.idyll` files in Idyll-MR. To view a story, the user runs the Idyll-MR server program with the `.idyll` file. The server program provides entry URLs for multiple types of viewing environments, including 2D screens (with a web browser), VR headsets (with a WebVR-enabled web browser), and AR headsets (specifically, HoloLens with a custom rendering application built on top of HoloJS).

2D vs. 3D content Idyll-MR supports both 2D and 3D components. For its 2D components, Idyll-MR currently supports a subset of Idyll's built-in components (see Table 8.1; remaining ones can be easily implemented in the future). Idyll-MR's 3D AR/VR components appear only in the AR/VR viewing mode.

Authoring vs. Viewing The web environment provides a natural vertical layout flow, which is not readily available in immersive environments. Authoring in immersive environments requires the careful placement of components in the 3D space. While Idyll-MR provides several layout constructs to ease the placement of components, and more automatic placement options are planned, manual placement is still essential. Therefore, in addition to the declarative `.idyll` file, Idyll-MR provides an interactive editing mode that features a user interface for manual layout. In the editing mode, we allow authors to move components in 3D space and resize them until the desired placement is reached. In viewing mode, all editing widgets are disabled to not interfere with the viewing process.

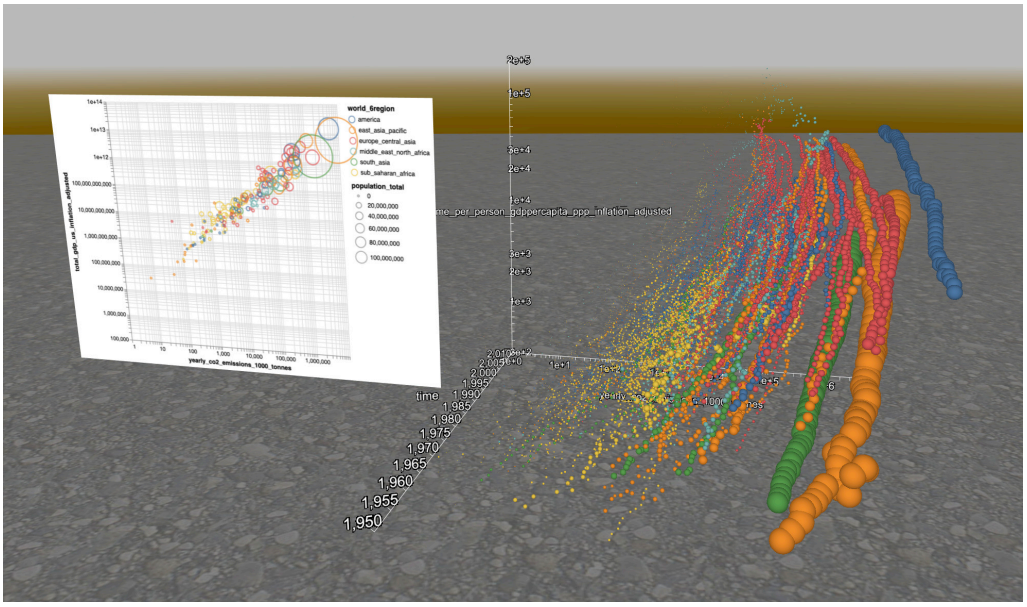
Idyll Components	2D Visualizations	Immersive Components			
		Static	Interactive	Visualization	Layout
`Display`, `Button`, `Slider`, `Interval`, `Radio`, `Select`, `Timer`, `Multiselect`, `Conditional`	`VegaSimple2D`, `Vega2D`, `VegaLite2D`, `Scatterplot2D`	`Text3D`, `Image3D`, `OBJModel`, `PanoramaBackdrop`, `GroundPlane`	`Button3D`, `Slider3D`, `Interval3D`, `Multiselect3D`	`VegaSimple3D`, `Vega3D`, `VegaLite3D`, `Scatterplot3D`, `CustomComponent3D`	`Grid3D`, `HeadUpDisplay`, `LODDisplay`

Table 8.1: Currently implemented Idyll-MR components

8.2.3 Mixed Reality Components

Many overall concepts for visual data stories [3] are still relevant in immersive environments, including the use of story pieces that are supported by and automatically linked to data, the use of visual representations with annotations or narrative to convey the message, and the connection of story pieces (*i.e.*, narrative flow [212]) in spatial or temporal order to support the overall presentation goal. Below, we discuss specific new elements for *immersive* data-driven stories in terms of visual representation, layout, and connections. We discuss the Idyll-MR components for each of these aspects. Table 8.1 shows a comprehensive list of components in Idyll-MR, including original Idyll components, additional 2D visualizations, and immersive components (split up into static, interactive, visualization, and layout components).

Immersive Visualizations In web-based or desktop-based visual data stories, data visualizations are usually limited to the 2D space. 3D visualizations can be used, but they require special consideration such as navigation controls. In immersive environments, 3D visualizations can be perceived better thanks to users' inherent navigational capabilities. In addition, stereo displays are commonly used in virtual and augmented reality devices, allowing better depth perception. However, even with better navigational capabilities and depth perception, 3D visualizations may still be harder to perceive. Therefore,

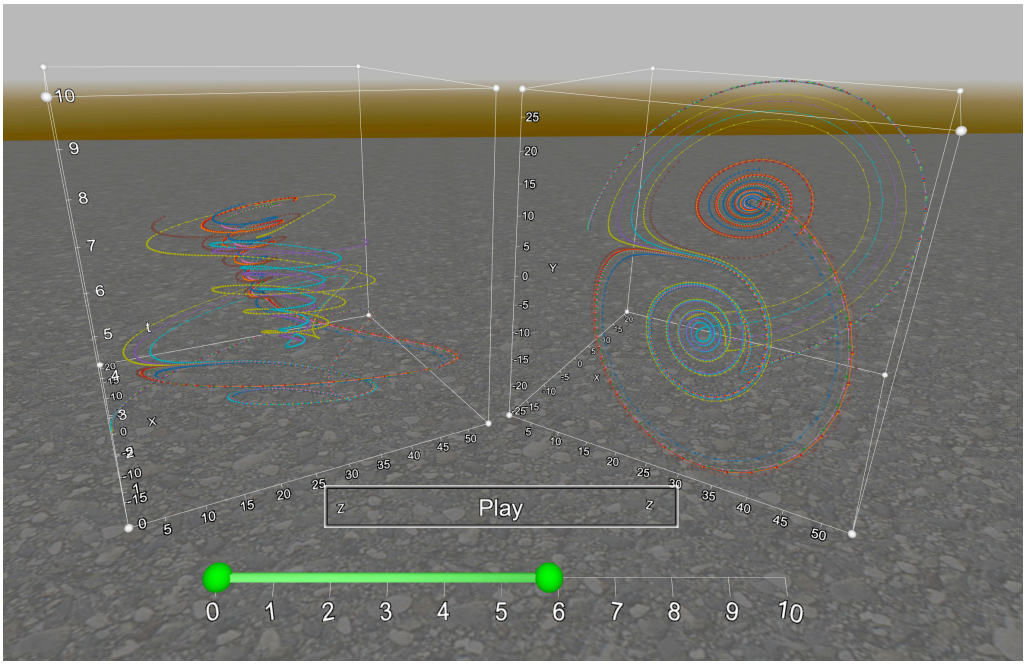


(a) A grid layout placing a Vega bubblechart and a 3D one side by side.

```
[Grid3D x:2]
  [VegaSimple3D src:"..."
    mark:"point"
    x:"yearly_co2_emissions:Q:log"
    y:"total_gdp:Q:log"
    size:"population_total:Q"
    color:"world_6region:N"
    filter:"datum.time == 2010"
  /]
  [Scatterplot3D src:"..."
    x:"yearly_co2_emissions:Q:log"
    y:"income_per_person:Q:log"
    z:"time:Q:domain(2014,1950)"
    size:"population_total:Q"
    color:"world_6region:N"
    linkBy:"geo" label:"name"
  /]
[/Grid3D]
```

(b) The Idyll-MR code to create this layout

Figure 8.1: Idyll-MR layout and visualization example.



(a) A head-up display showing a "play" button and a interval slider. The scene consists of two 3D scatterplots.

```
[var name:"play" value:false /]
[Timer
  trigger:play
  value:timeMax
  start:0 end:10 step:0.01 interval:0.1
/]
[HeadUpDisplay]
  [Button3D
    pose:`{y:-0.3,z:-1}`
    scale:0.5 sizeX:0.5 sizeY:0.06
    text:"Play" onClick:`play = true`
  /]
  [Interval3D
    pose:`{y:-0.4,z:-1}` sizeX:0.8
    valueMax:timeMax valueMin:timeMin
    min:0 max:10 digits:2
  /]
[/HeadUpDisplay]
```

(b) The Idyll-MR code to create the components in head-up display mode.

Figure 8.2: Idyll-MR head-up display example.

we still consider 2D planar visualizations an important type of visual representation in immersive visual data stories.

Idyll-MR currently supports three types of immersive data visualization components. (1) 2D planar visualizations can be created with Vega [162] or Vega-Lite [37]. For example, the 2D scatterplot in Figure 8.1-a is specified with the Idyll-MR component `VegaSimple3D`. The `VegaSimple3D` component provides a concise syntax (inspired by the Altair [213] library) to write Vega-Lite specification. For example, a quantitative encoding that maps `column_name` to `x` can be written as `x:"column_name:Q:log"`. The author may still use the plain JSON format of Vega or Vega-Lite in the `Vega3D` or `VegaLite3D` components, respectively. (2) Idyll-MR includes a 3D scatterplot visualization with the component `Scatterplot3D` (see the 3D scatterplot in Figure 8.1-a as an example); (3) If existing components are not sufficient, advanced authors can create custom visualizations with JavaScript code using the `CustomVisualization` component. The custom visualization can make use of existing libraries such as Stardust [13] or Three.js [214]. All visualizations can load data from a remote data server. We use Vega's "signals" concept to pass variables to visualizations. One important usage of signals is as parameters for filters. Below is an example. This example shows three attributes from the Gapminder [215] dataset in a 3D scatterplot. A set of checkboxes is shown besides the scatterplot for the viewer to filter the scatterplot by world region (code unrelated to the filtering mechanism is omitted):

```
[var name:"regions" value:`[...]` /]
[Multiselect3D value:regions options:`[
  "america", "east_asia_pacific", ...
]` /]
[Scatterplot3D
  src:... x:... y:... z:...
  filter:
    "indexof(signals.regions, datum.world_6region) >= 0"
  signals:`{ regions: regions }`
/]
```

Immersive Interaction Existing techniques such as brushing & linking and dynamic queries can be readily incorporated into immersive visual data stories. However, the effectiveness of such interactions can be different from regular desktop environments. For example, the limited field of view that an AR device provides may not be sufficiently large to display two 3D scatterplots for brushing & linking. A viewer may not be able to see the second plot while interacting with the first one. As an authoring system, Idyll-MR provides the possibilities for such interactions so that the author can make design decisions.

In terms of interactions that control the narrative flow (*e.g.*, the "next slide" button in a slideshow), immersive environments provide more opportunities, such as teleport and "world within world". Teleport allows a viewer to jump to a distant location without having to physically walk there; "World within world" is a technique that changes the scale of the viewer (or equivalently, the world) so that the viewer feels s/he is a mini-person or a giant (depending on scaling up or down). Such techniques can be useful for articulating particular story pieces in immersive visual data stories.

Many input elements for 2D screens can be ported to immersive environments with 3D user interfaces. Our current prototype includes `Slider3D` and `Interval3D` components for number and interval input respectively (see Figure 8.2). These are useful for parameterizing numerical filters. Radio buttons and multiple checkboxes are provided with the `Radio3D` and `Multiselect3D` components respectively. These can be used for categorical filters. For single actions, Idyll-MR also includes a `Button3D` component. With the supported event handlers, a button can trigger scene transitions or animations (see Figure 8.2).

Immersive Layout Although some existing layout patterns can be extended into immersive environments with an additional depth dimension (*e.g.*, grid layout can be ex-

tended to a 3D grid layout), layout in immersive environments is drastically different from web-based or desktop-based environments: (1) In terms of coordinate reference frames, a visual element can be world-stabilized (attached to a fixed physical location), or head-stabilized (attached to the viewer's head, *i.e.*, heads-up display); (2) Point of view is important in immersive environments. It can be problematic if the viewer is looking at an object with a sub-optimal viewing angle. To address this, the "billboarding" technique can be used to make the object automatically face the viewer. Level of detail is another concern in addition to viewing angle. When an object is far from the viewer, its details are not necessarily visible (especially when current VR/AR devices have limited-resolution displays). As a result, the object should respond to the viewer's distance and decide the right amount of information to display.

In Idyll-MR, layouts are primarily supported by layout components, including: (1) `Grid3D`, which arranges the child components in a grid specified by the author (see Figure 8.1). The author can specify the number of components in the x, y, and z directions, and optionally specify their proportional size; (2) `HeadUpDisplay`, which attaches child components to the viewer's head (aka., using the view plane as the coordinate reference, see Figure 8.2), and (3) `LevelOfDetailDisplay`, which displays a different child component based on the viewer's distance (the threshold can be specified by the author). In addition to layout components, 2D visualizations, images, and buttons have an optional `billboarding` switch, which allows the author to choose between a fixed orientation or billboarding (causing the component to always face the viewer).

8.2.4 Co-located Authoring and Viewing

We designed Idyll-MR with co-located authoring and viewing in mind: by enabling VR and AR users to operate in the same physical space, we allow multiple story creators

to simultaneously access the story design interface and collaborate on the story design. The scene can be shared across multiple devices, currently including HoloLens, HTC Vive, and desktop computers. We believe that collaboration among story creators with different devices can best utilize the devices' strengths and overcome the weaknesses. For example, a story creator with the desktop interface can serve as the visualization creator, who maintains the individual visualizations. A story creator with a see-through AR device has access to both the desktop interface and the immersive space in 3D, and is therefore able to optimize the spatial layout of the visualizations to best suit the available physical space. A story creator with a VR device may experience the designed scene and report the results or any issues to the other two creators.

To enable VR and AR users to efficiently collaborate, and for each to use the tracked VR controller as their interaction device, we offer procedures to calibrate VR and HoloLens tracking to each other. The HoloLens is being tracked using its own inside-out visual tracking system using World Anchors. The HTC Vive is using its proprietary laser-based inside-out positional tracking using multiple "lighthouses" that flood the environment with imperceptible structured light.

The calibration between these two tracking systems is done once after a new room-scale play area is set up for the VR device. Our calibration procedure currently asks the HoloLens user to align the Vive controller to a virtual model of controller that is displayed in six different locations in sequence. From these correspondences, we calculate the correct transformation matrix between the two coordinate systems and then can operate in the same physical space. The HoloLens tracking system being more flexible and wide-area, not relying on the light signals emitted by the lighthouses, an AR user can observe the virtual scene even from outside the Vive tracking area. However, to utilize the Vive controller as an input device, that controller needs to be within the VR play area. HoloLens tracking with world anchors is reasonably reliable, but not as resistant to

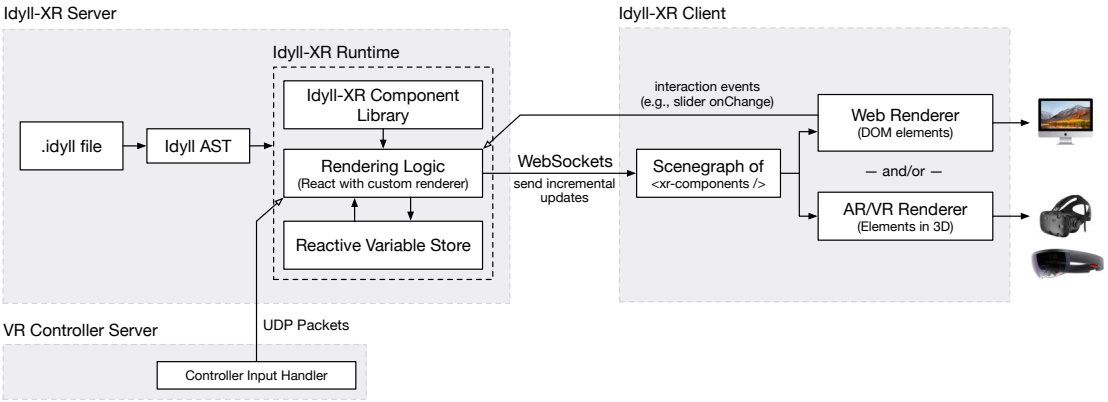


Figure 8.3: The architecture of Idyll-MR.

drift and tracking loss as the lighthouse-based Vive solution. Should HoloLens tracking deteriorate over time, a new calibration needs to be performed.

8.2.5 Implementation

Idyll-MR enables co-located authoring and consumption of immersive data-driven stories across multiple devices. We decided to use a server-client architecture (see Figure 8.3). The *Idyll-MR Server* maintains the story, performs the application logic, synchronizes the displayed content with the AR/VR/Desktop devices, and receives user input from the VR controllers, devices, as well as commands from the desktop user interface; the *Idyll-MR Clients* support rendering on multiple environments, including web, virtual reality, and augmented reality. In particular, we currently support modern web browser environment, WebVR-enabled browsers for VR devices (*e.g.*, HTC Vive's Google Chrome browser), and the HoloLens with a custom rendering application based on the HoloJS [216] library.

Incremental Scenegraph Synchronization To facilitate the easy implementation of the system, we use React [181] to manage the rendering scenegraph for desktop and immersive environments. This allows us to componentize the system and use the Flux [183]

architecture that consists of a store, a set of components, and a dispatcher to implement the application logic.

However, the default React library targets the HTML DOM, which is not designed for immersive applications. More importantly, it is not trivial to synchronize the DOM tree with rendering clients. In order to address the former issue, we define a set of special `<xr-components />` as the building blocks of the rendering scenegraph; to address the latter issue, we build a custom rendering backend for React, which receives modification commands from React, and converts them into incremental updates to the scenegraph. The incremental updates are then sent to rendering clients via the network, and applied to the clients' scenegraph upon arrival. With this method, we have the benefit of React, a well-established component framework, and the reduced communication cost from incremental synchronization.

The abovementioned approach incremental scenegraph synchronization facilities can be used to implement a variety of applications. On top of this, Idyll-MR implements the support for handling and rendering Idyll components. To do so, we followed Idyll's original approach which uses a reactive variable store to manage the variables in an interactive document, and use the React library with our custom backend to render the components.

Interaction Our implementation allows two types of interaction inputs: (1) widget events from the desktop computer (*e.g.*, dragging a range slider); and (2) VR controller inputs. Events from desktop widgets are sent to the corresponding React component through the network. The state of the VR controllers and their button events are forwarded to the server by a dedicated process that communicates with the VR system (built on top of OpenVR). The server processes the controller states and their button events, and dispatch them to the appropriate React components. As shown above, the Re-

act components receive the final events from both types of inputs. These components are responsible for interpreting the inputs and produce the right response. For example, the author can map a slider's value to an Idyll variable. Then the component is responsible to receive the value from the input widgets, and update the Idyll variable accordingly. In the case of a 3D slider, the component is responsible for translating controller poses into correct slider values. To ease development and to make the user interface more consistent, we implement common types of 3D interactions, such as dragging with controller, as reusable React components. Higher-level components, such as the `Scatterplot3D` visualization, can then readily incorporate these interactions.

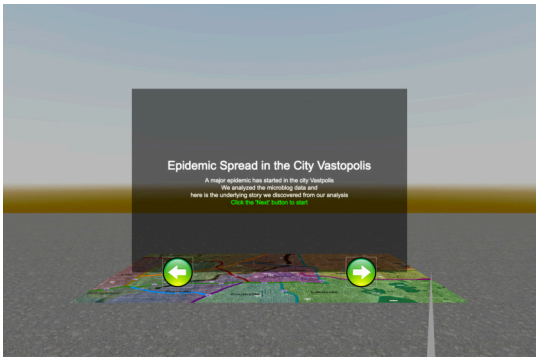
8.3 Examples

We illustrate the possibilities of Idyll-MR with three examples. Together they demonstrate Idyll-MR's expressiveness and ease of coding for creating *immersive* visual data stories.

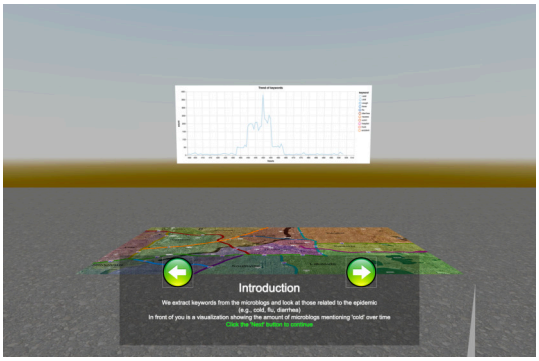
8.3.1 Ex1: Characterization of an Epidemic Spread

Our first use case illustrates the authoring of a simple immersive visual data story (Figure 8.4). Here, we use the data from the 2011 IEEE VAST Challenge [217] and present data from mini challenge 1, *Characterization of an Epidemic Spread*. With Idyll-MR, we have put together a simple sequence of scenes that get triggered via button-based transitions.

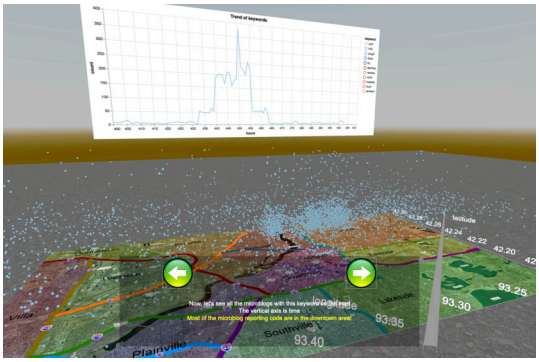
Figure 8.4-a shows some introductory text to illustrate the background of the event. A pair of previous and next buttons are shown below the text for the viewer to navigate the story. The text and buttons are placed in the head-up display to make it easy for the viewer to find them. Once the next button is pressed (via the VR controller), the story transitiosn into the second scene (Figure 8.4-b). In this scene, we place additional



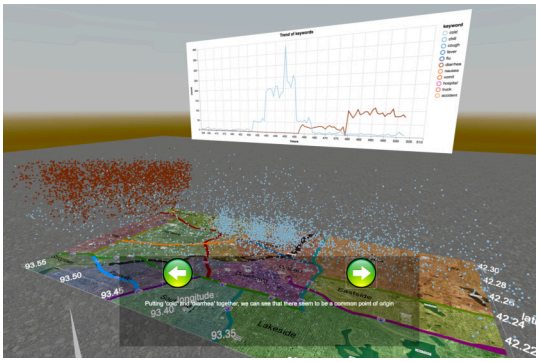
(a) Title and introduction.



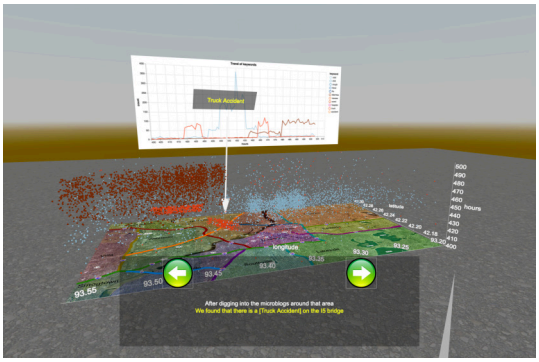
(b) A single 2D chart for the keyword "cold."



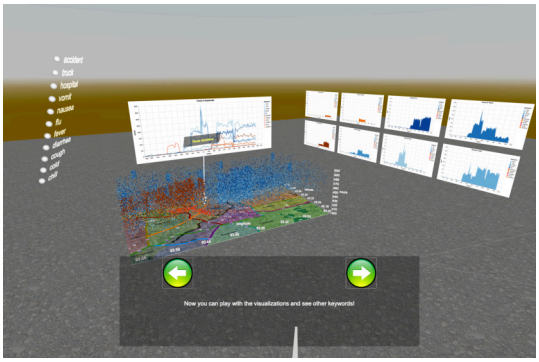
(c) 3D scatterplot showing the location and time for microblogs with the keyword.



(d) Add another keyword: "diaherra."



(e) Show microblogs related to "truck" to illustrate the root cause.



(f) Show all keywords and interactive widgets to allow the viewer to explore the scene.

Figure 8.4: Example immersive visual data story created with Idyll-MR, telling the story of the IEEE VAST Challenge 2011 (mini challenge 1). This story consists of multiple scenes controlled by a set of navigation (previous, back) buttons. It introduces the keywords and events progressively, leading to the key insight. In the end, the viewer is given freedom to explore the dataset with an interactive filter.

textual introduction in the head-up display below the buttons, and have a 2D visualization appear above the map of the city "Vastopolis." In the 2D visualization, we can see that the frequency of the word "cold" spikes at a certain time. This is the first sign of the epidemic spread. The next scene, Figure 8.4-c, adds the microblogs with the word "cold" on top of the map. From this 3D visualization, we can see that the microblogs are centered around the downtown area. The next scene, Figure 8.4-d shows an additional keyword "diaherra," which has a clearly different spatial and temporal pattern (time is shown in the vertical axis). These tweets are located around the VAST river, and happens the day after the "cold" microblogs. In the next scene, Figure 8.4-e, we illustrate the cause of this epidemic, which is a truck accident on the bridge over the VAST river. The truck contains pathogens that causes these two types of disease. This fact is illustrated by the red dots that indicates microblogs with the word "truck," and an arrow and text label to clearly show it's a truck accident event. Finally, in Figure 8.4-f, we show all microblogs with a richer set of keywords (*e.g.*, fever, flu, hospital), and allow the viewer to interactively explore the scene through a checkbox list (shown at the right). The scene also features a grid of 8 2D visualizations showing the timelines of each keyword separately.

The story uses the `Image3D` component to show the map, the `Scatterplot3D` component for the 3D visualization, and `VegaSimple3D` components to show the 2D visualizations. The `HeadUpDisplay` is used for the buttons and text. The current scene is recorded in a numerical variable `current_scene`, and is updated by the buttons' `onClick` event. The content of the text view, the visible microblogs in the 3D visualization, and the visible timeline in the 2D timeline view are all conditioned on the `current_scene` variable.

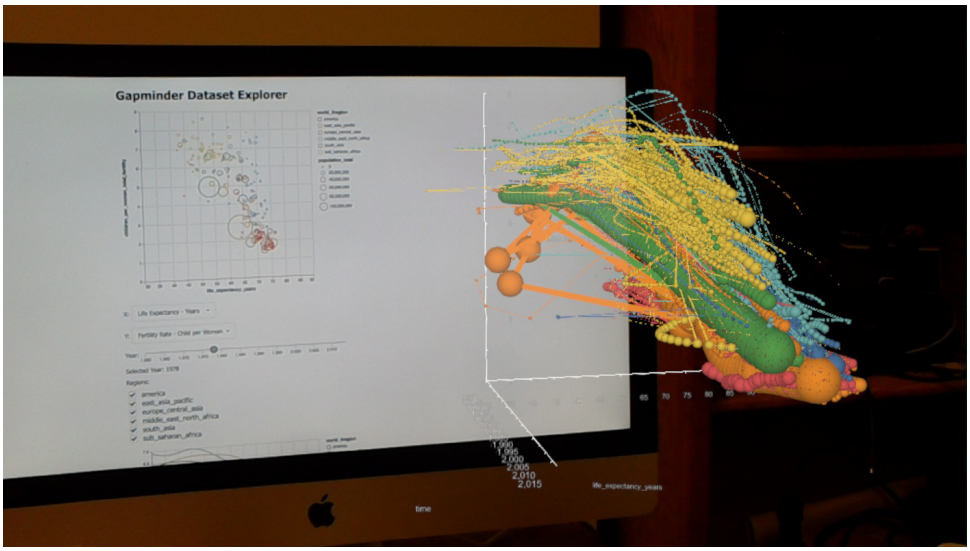


Figure 8.5: A view from the HoloLens of a Gapminder explorable example created in Idyll-MR. The 3D scatterplot visualization in the HoloLens shows two viewer-selected indicators and time, and the 2D scatterplot in the monitor shows a scatterplot of the two indicators, with a viewer-adjustable time.

8.3.2 Ex2: Gapminder Explorable

In our second example, we use an AR device (HoloLens) to augment a 2D display, illustrating a different form of immersive visual data story. As shown in Figure 8.5, we first create a 2D explorable visualization, mimicking the original Gapminder tool [215]. It features a bubble chart showing two indicators (*i.e.*, attributes) at a given year, and the size of bubble represents the total population of the corresponding country. The viewer can decide which year to look at with the "year" slider, and select the two indicators with a drop-down selection widget. With Idyll-MR, we augment this 2D component with a 3D scatterplot that shows the two indicators with time as an additional axis. This scatterplot may reveal interesting 3D patterns that are otherwise hard to see by just looking at and interacting with the 2D view. In terms of interaction, brushing and linking is enabled between the two plots (see the supplemental video), and this component also features a list of checkboxes that allows the viewer to filter the countries by region.

Note that Idyll-MR uses its own implementation of Idyll components, and they look different from the original Idyll components. The 2D/3D scatterplots are created using Idyll-MR's `Scatterplot2D` and `Scatterplot3D` component, respectively, while the remaining part shown in the 2D display is implemented with Idyll components.

8.3.3 Ex3: Lorenz Attractor

A good use case of the 3D scatterplot is to visualize structures that are inherently 3D. In this example, we show the well-known Lorenz attractor in virtual or augmented reality. The dataset contains a simulation of the Lorenz equation with 10 starting points. Since the Lorenz equation is chaotic, the trajectories of these points eventually diverge from each other significantly. Figure 8.2 shows a view of this example in virtual reality. It consists of two 3D scatterplots: a (x, y, z) scatterplot that illustrates the overall 3D structure of the Lorenz attractor, and a (x, y, t) scatterplot showing two spatial dimensions with respect to time. We enable brushing and linking between the two scatterplots so the viewer can correspond them more easily. In order to better illustrate dynamic processes, we add a "Play" button that triggers an animation showing the trajectories gradually. Furthermore, we have a interval slider below the play button to allow the viewer to filter by time interval when the animation is not running.

The brushing and linking interaction is specified by assigning a shared `selection` variable to the two 3D scatterplots. The interval slider controls a pair of `tMin`, `tMax` variables, which are used for filtering the data points in the two scatterplots. Finally, the animation is implemented using Idyll-MR's `Timer` component, which can be triggered by a button press.

8.4 Limitations and Future Work

Components and Extensibility While we have discussed our choices of AR/VR components, Idyll-MR currently only have built-in support for a subset of such components. For example, it currently supports only one 3D data visualization: 3D scatterplot. Incorporating a grammar for additional 3D visualizations is left for future work. In terms of extensibility, adding new components to Idyll-MR requires modifying its codebase directly, both the server side and the rendering client's side. Providing an API to import additional components is left for future work.

Authoring Options In the current prototype of Idyll-MR, we implemented textual authoring via the Idyll language, and manual layout options (with the help of some layout components). Further improving the 3D interaction techniques that Idyll-MR offers, which are currently simple ray-based interactions, executed with Vive controllers is left for future work. We purposefully kept the controller mapping simple, utilizing mainly the trackpad for interactions along the pointing ray. Additional interaction mechanisms for exact manipulation at a distance, and especially constraint-based placements are needed. Currently, the real-time updated mesh from HoloLens mapping is not taken into account for object placement and alignment. This is an immediate next step for future work. When more authoring functionality is added to the system, we need to provide more sophisticated menu structures around the controller, including self-explaining tutorial overlays, as are now fairly common in VR apps. We can do this in both VR and AR since our HoloLens applications use the same controller interface, as long as it shares the VR tracking space.

Performance & Latency Our system performs smoothly on web and PC-driven virtual reality environments for all our examples. It also works well on mobile VR such as the

standalone Lenovo Mirage headset. However, HoloLens performance somewhat suffers from a slow initialization time and a lower framerate (as revealed in the supplemental video). When the story becomes complex, these issues renders the application difficult to use in the HoloLens. This is due to the fact that the HoloJS [216] library we are using lacks advanced pipeline optimizations that modern web browsers support. In terms of latency, we found that through a 1Gbps Ethernet connection, the WebVR version works well for the scenes shown in this chapter. The HoloLens implementation suffers from a bit more latency. Thanks to the fact that the application state is kept entirely in the server, when the application in the HoloLens disconnects or crashes, the user can restore it by a simple restart without loss of information.

Evaluation We plan to conduct user studies to evaluate Idyll-MR's usability. We would particularly like to formally evaluate if multiple story creators can effectively and gainfully collaborate using Idyll-MR. While formal usability evaluation of Idyll-MR is left to future work, we have gathered extensive anecdotal evidence for the usefulness of a collaborative design and viewing process. The different roles of authors include: web-based authors and viewers, AR authors and viewers, and VR authors and viewers. The system can theoretically support arbitrary numbers of each of these roles, but we have only tested the system with one Vive user, one HoloLens user, and multiple web users at the same time. Likewise, other VR headsets that support WebVR could easily be supported by Idyll-MR, but the unified tracking and interaction would require some additional support.

In practice, the pros and cons of the HoloLens and Vive systems we integrated into the system make for an interesting division of labor in authoring immersive data-driven stories. The person who authors visualization components using the desktop/web front end can at the same time utilize the (relatively lightweight) HoloLens AR headset to test the appearance of their visualizations immediately in the immersive space. Thus, the

roles of web author and AR author often conflate. The AR headset is of use even when designing an immersive data-driven story that is optimized for VR consumption, because, again it is the more mobile and conveniently wearable device, and also, the AR author can take into account idiosyncracies of the physical-space VR tracking area, for example, making sure that the VR user will not bump into desks or chairs when examining certain virtual content. On the other hand, the small HoloLens field of view (about $30\times17^\circ$) makes it very difficult to view and explore large nearby immersive visualizations conveniently. The VR modality is much better suited for expansive immersive data spaces, and to get an overview of any new data story.

8.5 Conclusion

In this chapter, we have introduced Idyll-MR, an authoring language and system for easy creation of immersive visual data stories, and discussed its design and implementation. Idyll-MR extends the existing Idyll language by adding components for immersive visualizations, interaction, and layouts. We have demonstrated the expressiveness of Idyll-MR through example immersive visual data stories. We have also discussed its limitations and explored opportunities for future work. We hope the Idyll-MR system will help VR and AR users create powerful immersive data visualizations and stories, and inspire researchers to expand on this work and design and evaluate systems for immersive data-driven storytelling.

Part III

Discussion and Conclusions

In this part, we first present a discussion on the evaluation methodologies of interactive visualization authoring systems. We reflect on our own experiences in evaluating the visualization authoring systems that we have developed as well as the evaluation methods used in other recent projects. We also examine alternative approaches for evaluating visualization authoring systems that we believe to be more appropriate than traditional comparative studies. Based on our reflection, we present opportunities for facilitating the evaluation and adoption of deployed visualization authoring systems.

After this discussion of evaluation methodologies, we discuss other general issues and limitations pertaining to the visualization authoring systems we have built and point out future research directions. We also extend our discussion beyond visualization authoring tools, such as the incorporation of data wrangling functionalities and the integration into presentation tools. Finally, we recapitulate and conclude the dissertation.

Chapter 9

Reflecting on the Evaluation of Visualization Authoring Systems

9.1 Introduction

With an increasing demand for data-driven storytelling, we are witnessing a proliferation of visualization authoring systems [49, 219, 220, 80]. Although these systems pursue a similar goal (*i.e.*, enabling people to easily visualize data), it is seldom straightforward to understand and assess a novel authoring system's strengths and weaknesses relative to other systems. In this chapter, we discuss more appropriate ways to evaluate visualization authoring systems.

While the difficulties and challenges in evaluating information visualization systems have been discussed at length in the visualization research community [221, 222, 223, 224], the evaluation of visualization authoring systems presents additional unique challenges,

The contents of this chapter have been previously published in *Proceedings of BELIV 2018: Evaluation and Beyond – Methodological Approaches for Visualization.* © 2018 IEEE. Reprinted, with permission from Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche, *Reflecting on the Evaluation of Visualization Authoring Systems : Position Paper*, *Proceedings of BELIV 2018: Evaluation and Beyond – Methodological Approaches for Visualization* [14], Oct. 2018.

Figure 9.1: A selection of visualization authoring systems developed by one or more of the authors. From left to right: iVisDesigner [10], DataClips [70], ChartAccent [11], Timeline Storyteller [218], DataInk [189], and Charticulator [12].

calling for alternative evaluation approaches.

Research papers describing visualization authoring systems may offer different research contributions: they may propose a new approach to visualization authoring, target a new group of prospective visualization authors, enable the creation of custom visualization designs, or any combination of these. Typical evaluation metrics employed by visualization researchers may not be suitable for evaluating these authoring systems. For instance, while efficiency and usability may be relevant in cases where the aim of the system is to improve an existing authoring workflow, these metrics may be irrelevant if the system's goal is to offer new levels of expressiveness to visualization authors.

It is challenging but critical to clearly define the specific contributions of a novel visualization authoring system, as these will guide the evaluation criteria and the approach to assessing the results of these evaluations on the part of reviewers. Being explicit about this evaluation rationale is critical for dispelling unrealistic expectations that the system should "do it all": be powerful yet easy to learn, suit novice and expert audiences alike, and outperform other authoring systems in terms of expressiveness, usability, and efficiency. In this chapter, we reflect on our own experience in evaluating the visualization authoring systems that we have developed in recent years to facilitate visualization designs for communication purposes. Our intent is to provide insights to both researchers

and reviewers with regards to different types of contributions, evaluation criteria, and evaluation methodologies. We conclude by discussing opportunities for the visualization community to better facilitate the evaluation of visualization authoring systems.

9.2 Challenges

Amini *et al.* [225] recently identified seven criteria relating to the evaluation of tools for authoring data-driven stories. We build upon and tailor this list of criteria to visualization authoring systems, integrating insights from our own experience. Note that this list is not exhaustive and may grow as more visualization authoring systems emerge in the future.

- *Expressiveness:* The scope of possible visualization design choices enabled by the system.
- *Creativity support:* The extent to which a system aids the author in creating novel visualization designs, such as easy manipulation of graphical elements or easy specification of element layouts.
- *Flexibility:* The number of ways an author can achieve a desired visualization design.
- *Guidance:* The extent to which an author can produce a visualization without external assistance.
- *Efficiency:* How quickly a desired visualization design can be produced using the interface, how many actions are required to produce the desired design, or how many visualization design choices can be made in a set amount of time.
- *Usability:* How easily a desired visualization can be produced using the system.

- *Learnability:* The ease of learning and recalling interactions within the system after initial guidance.

- *Integration:* The extent to which the system fits into an authors' workflow. This may include supporting specific sequences of tasks, bridging to existing tools, or supporting collaboration.

Traditional controlled experiments are useful to compare efficiency (*e.g.*, task time, error rate). However, they are not always appropriate for evaluating other criteria, particularly in the context of visualization authoring systems, where it is rare to find an appropriate baseline to compare a new system against. Researchers often design and develop systems because existing systems are not designed to support desired capabilities. It is also unrealistic to have a study session of sufficient length such that participants learn the full capabilities of a visualization authoring system. Consider, for instance, the number of features in commercial software tools such as Adobe Illustrator, which many people use during the visualization authoring process. Therefore, it is difficult for researchers to control important factors or to select tasks without penalizing one of the systems or compromising the external validity of the study.

Suitable evaluation methods should be selected based on the evaluation criteria, which in turn should depend upon the research contribution that the researchers intend to make. For example, if the contribution is a system that allows authors to create a wider variety of visualization designs (*e.g.*, Lyra [8]), *expressiveness* may be the primary evaluation criterion. If the system's contribution is bridging the strengths of multiple authoring tools (*e.g.*, Hanpuku [82]), the evaluation should be centered around assessing the degree of *integration*. When an authoring system enables people to produce artifacts that may be novel or understudied, independent studies on the evaluation of the artifacts may be necessary. Note that the researchers who develop such systems may address several of

these criteria, and sometimes they need to strike a balance between them. For example, if a system targets novices (*e.g.*, ManyEyes [58]), it may provide a high level of *guidance* at the expense of *flexibility* (requiring only a limited number of simple steps to author a visualization) and *expressiveness* (providing a constrained set of visualization design choices). Finally, we must also consider the *usability* and *learnability* for the target audience. These criteria are easier to satisfy when targeting a narrow audience such as professional designers or graphic journalists, as opposed to targeting the general public.

Several evaluation methods lend themselves to a single criterion. For example, a gallery of visualization designs would be appropriate for demonstrating *expressiveness*, while a traditional usability study would be suitable for validating the usability of the authoring system. However, it is not always trivial or apparent to pick appropriate methods for certain criteria. In such cases, the researchers should justify their choice of evaluation methods.

9.3 Reflection on Existing Approaches

The authors of this chapter have collectively designed, developed, and evaluated a set of visualization authoring systems (see Figure 9.1). In this section, we reflect upon the experience of evaluating these systems. We also discuss methods used to evaluate other recent visualization authoring systems whose main goal was to support "expressive" visualization design. Table 9.1 summarizes the evaluation methods used for each of the systems. We identify the strengths and weaknesses of each evaluation method in the hope that researchers can choose appropriate methods to evaluate their own systems.

	Formative Study	Reproduction Study	Free-Form Study	Comparative Study	Gallery
ChartAccent [11]	survey[1]	✓	-	-	-
Charticulator [12]	-	✓	-	interaction complexity	🖼 🎥
DataClips [70]	survey[1]	-	✓	video design	-
DataInk [189]	-	✓	✓	-	-
iVisDesigner [10]	-	-	informal	-	🖼
Timeline Storyteller [218]	survey [71][1]	-	-	-	🖼
Data Illustrator [67]	design meetings	✓	-	-	🖼 🎥
Ellipsis [226]	interviews	-	✓	-	🖼
Hanpuku [82]	-	-	-	-	three examples
Visualization-by-Sketching [227]	short sessions of demos and discussions	-	-	-	five examples
InfoNice [114]	-	-	✓	time	🖼
iVoLVER [163]	-	-	-	follow-up research [228][2]	🖼 🎥
Lyra [8]	-	✓	-	-	🖼
VisComposer [68]	-	✓	-	time	🖼

Table 9.1: Evaluation methods used for each of the visualization authoring systems. 🖼: gallery of visualizations; 🎥: video showing the creation process. [1]These surveys of existing artifacts helped us extract a design space. [2]iVoLVER was evaluated against Tableau in a follow-up study [228].

9.3.1 Formative Study

Brehmer *et al.* [229] encourage visualization researchers and practitioners to conduct pre-design empirical studies, as they can inform system design through the characterization of work practices and associated problems. In developing Data Illustrator [67], Liu *et al.* held weekly meetings with three designers over the span of two years as a means to better understand how visualization could be described and produced from a graphic design perspective. Similarly, Schroeder and Keefe sought feedback from professors and students (*i.e.*, the system's target audience) over the course of two years through short sessions of demonstrations and discussions, which helped them shape the interface design [227]. We note that these are exemplary instances of a formative or pre-design study.

Another way to establish the desired expressiveness of a novel authoring system is to review a collection of existing artifacts. For example, in designing DataClips [70], an authoring system for data videos, Amini *et al.* first examined 50 data videos collected from various media sources online in an effort to tease apart the design dimensions used to produce compelling narratives in film or cinematography [230]. Similarly, Ren *et al.* surveyed 106 existing annotated charts published by several news media outlets to generate a design space of annotations, which subsequently informed the design of ChartAccent [11]. Finally, Brehmer *et al.* designed Timeline Storyteller [218] based on a recently proposed design space grounded in an extensive survey [71].

A formative study can be used to justify an authoring system's design, such as the underlying visualization design framework or its interaction mechanisms, how the interface will integrate with existing work practices and tools, how skilled users of existing systems may transfer their skill to the new system, and whether and how multiple people collaborate during the visualization design process. For example, the partition and repeti-

tion actions in Data Illustrator [67] were attributed to findings from their formative study and observations of how designers use existing tools such as Adobe Experience Design. A formative study can also inform design trade-offs, such as between expressiveness and interface complexity; for example, ChartAccent [11] prioritizes the design of the most common forms of annotation, while some of the more esoteric forms of annotation found during the formative survey are not supported.

9.3.2 Reproduction Study

The focus of a reproduction study is the usability of a system and the learnability of its features. The evaluation of Lyra [8], ChartAccent [11], VisComposer [68], DataInk [189], Data Illustrator [67], and Charticulator [12] each featured a *reproduction study*, in which study participants were asked to reproduce a copy of one or more visualization designs.

A reproduction study typically employs a think-aloud approach, which helps researchers elicit the subjective impressions of participants as they use the system for visualization design tasks. Visualization design completion time can only serve as a proxy of efficiency, as participants are discovering the interface for the first time and explaining their thought process as per the think-aloud protocol; both of which can inflate the time to reproduce a design. As the study participants show different think-aloud behavior, the variance in time measurement is likely to be substantial: some participants provide more detailed explanations than others. However, researchers can still assess if participants can produce a desired visualization design, and if not, what the main barriers might be. The think-aloud protocol can also be complemented with experimenter observations, screen capture video, and post-study questionnaires.

Indirectly, a reproduction study may also shed light on whether participants understand the interaction or design framework embodied by the authoring system. For

instance, Charticulator [12] incorporates a novel constraint-based chart layout design framework that separates mark and glyph construction from chart layout. By observing and listening to participants think aloud as they reproduce a set of designs with Charticulator, researchers could infer whether (and when) they understood the effects of layout constraints based on their interactions and utterances.

It should be noted that, in practice, reproduction study sessions are brief (*e.g.*, 60–90 minutes), and thus participants may learn and use only a subset of the system's features or capabilities. Similarly, any tutorial they receive prior to using the system may only address a subset of features; ideally this subset includes those which underlie the research contributions of the system.

Reproduction studies of research prototypes are likely to be hindered by minor but common usability issues. For instance, in Data Illustrator's reproduction study [67], participants remarked upon the lack of undo/redo functionality; similarly, participants commented on the lack of z-order manipulation in ChartAccent [11]. Such usability issues are often tangential to the system's research contributions, but they can impact its usability and by extension the quality of results collected during a reproduction study. Researchers should be aware of the considerable amount of effort required to address such issues so as to conduct a successful reproduction study.

Arguably, a reproduction study will not surface all of an authoring system's usability issues, and thus additional measures of usability should be undertaken should the researchers intend to deploy a usable tool. Unfortunately, findings from additional usability studies are unlikely to form substantial visualization research contributions.

9.3.3 Free-Form Study

In a free-form study, participants are asked to create their own visualization designs using the system. This method was used in the evaluation of Ellipsis [226], DataClips [70], DataInk [189], and InfoNice [114]. Such studies focus on assessing if participants can create visualizations of their own imagining using the new authoring system. Thus, they may include a design phase during which participants are asked to envision a visualization design before trying to realize it using the authoring system. For example, Amini *et al.* [70] included an *idea generation and sketching* phase before the creation phase. During the creation process, the experimenter may aid the participants by providing reference materials, such as a cheat sheet. In addition, a free-form study can be preceded by a reproduction study (*e.g.*, DataInk [189]), as it can serve as an additional tutorial.

Free-form studies are more externally valid than reproduction studies, resembling the real-world usage scenario of a visualization authoring system. They enable researchers to capture the learnability and usability of the system, as participants need to identify how to execute their own design. They may also capture additional insights relating to other criteria; for example, a free-form study with DataInk [189] enabled the researchers to capture evidence of creativity support by identifying the number of different designs that participants created, and by comparing the originality of these designs to existing ones.

Free-form studies require a relatively low degree of effort to execute and may provide useful insights that speak to evaluation criteria other than those targeted by reproduction studies, such as creativity support or expressiveness. On the other hand, they tend to be of short duration and occur in a laboratory setting. Therefore, they may not be appropriate for demonstrating the expressiveness of a system equipped with many features, which will require more time to learn and master. In these cases, other forms of evaluation are

preferred, such as design galleries (Section 9.3.5).

9.3.4 Comparative Study

Several researchers have conducted a comparative study to compare their authoring system against existing commercial software tools. For example, Amini *et al.* [70] compared the experience of producing data videos with their DataClips prototype against the combination of Adobe Illustrator and Adobe After Effects, tools that are commonly used in conjunction to produce data videos. They compared the time that study participants took to produce each video clip as well as the number of clips they created. Similarly, Méndez *et al.* [228] conducted a study comparing iVoLVER [163] against Tableau to better understand the authoring process across different tools. We also note that comparison can be done without recruiting human subjects. For example, Ren *et al.* compared Charticulator with three existing visualization authoring systems in terms of the number of interactions required to produce an equivalent visualization design, a proxy assessment of efficiency and of their respective complexity, inspired by the keystroke-level model [186].

While a comparative study such as a controlled experiment with quantitative metrics may appear to be the most objective way to compare multiple approaches, we find it particularly difficult to yield scientifically meaningful results due to many confounding factors. Existing visualization authoring systems differ not only in their interaction but also in their underlying design frameworks.They also differ in terms of the size and robustness of their feature sets; we have remarked above that many research prototypes prioritize the implementation of novel research features at the expense of or instead of seemingly mundane yet often-used features such as undo/redo and z-ordering. Even when researchers discover quantitative differences in completion time or interaction

count in a comparative study, it is very difficult to determine which design choices or combination of features yielded the differences, and accordingly the findings are difficult to generalize. Finally, an absence of basic features or the presence of minor usability issues present in a research prototype can easily be addressed soon after the study is conducted (*e.g.*, Data Illustrator implemented undo/redo after the completion of their study). This will often render obsolete the findings from preceding comparative studies.

9.3.5 Gallery

Recent visualization authoring systems such as DataInk [189], VisComposer [68], InfoNice [114], Data Illustrator [67], and Charticulator [12] are specifically intended for expressive visualization design. This expressiveness cannot be captured through a reproduction study. Even a free-form study may only capture limited insights with respect to expressiveness, as it largely depends on the participants' creativity at the time of the study. To focus on the expressive potential of the authoring system rather than of the participants, researchers will provide a collection of diverse visualization content as a gallery, either within a research paper as a gallery figure and/or as supplemental material, such as within a supplemental project website. The latter can also include video demonstrations of the authoring process for each item (an approach taken in the Data Illustrator [67] and Charticulator [12] projects), which has the additional benefit of increasing the interface's learnability and thus the adoption of the system.

Notable benefits of evaluating an authoring system via a gallery are twofold. First, it is perhaps the best way to demonstrate the expressiveness of the system. Second, these galleries can serve as a benchmark, such that developers of future authoring systems could compare the expressive range of their new system in terms of the degree of overlap relative to the content of previous galleries. For example, Charticulator's gallery [12]

reproduces much of the content from Data Illustrator [67] gallery while also introducing new content that cannot be reproduced using Data illustrator.

However, it is important to consider that while the authors of an authoring system can create unique and complex visualization designs, the target users may struggle to produce such content. In other words, a gallery does not assess the usability or learnability of the system, nor does it in itself serve as creativity support for its target audience. For this reason, a gallery is often paired with at least one form of evaluation involving human subjects.

9.3.6 Combining Multiple Methods

Expressive visualization design is a complex and creative process, requiring a considerable amount of time and effort. To provide a comprehensive evaluation of systems to support such activity, researchers often incorporate many of the methods discussed above in conjunction, to reach a consensus regarding the overall benefits and drawbacks of a novel system, perhaps with reference to more than one of the evaluation criteria discussed above. For example, Liu *et al.* evaluated Data Illustrated by means of a formative study, a gallery, and a reproduction study, while Ren *et al.* evaluated Charticulator [12] by means of a gallery, a reproduction study, and a comparative study, Xia *et al.* evaluated DataInk [189] by asking participants to reproduce a visualization and then create their own custom visualization during a free-form study session, and Mei *et al.* combined a reproduction study and a comparative study to evaluate VisComposer [68].

When combined with other evaluation methods, a highly limited evaluation can still provide valuable insights. For example, Schroeder and Keefe deployed two versions of the system to one artist [227]. Even though this evaluation alone was not enough to validate the system, it still helped the researchers gather feedback from their target audience to

improve the system.

9.4 Opportunities

In this section, we discuss potential ways to facilitate the evaluation and adoption of visualization authoring systems.

9.4.1 Benchmark Repository

A benchmark is defined as "a standardized problem or test that serves as a basis for evaluation or comparison (as of computer system performance)" [231]. One way to improve and facilitate evaluation is to develop a benchmark, and thus Plaisant called for the creation of benchmark datasets and tasks for evaluating information visualization systems [221]. As a first step, Fekete and Plaisant organized the first InfoVis contest [232] to initiate the development of benchmarks and to establish a forum to promote evaluation methods. They also created the Information Visualization Benchmark Repository [233] to archive materials from this contest, which was replaced and extended by the Visual Analytics Benchmarks Repository [234] in 2006. More recently, we have seen a number of additional visualization data and technique repositories, such as vispubdata.org [235], KeyVis.org (VIS paper keywords) [236], and treevis.net [237]. Similarly, the visualization research community would benefit from such a repository with a specific focus on visualization authoring, which provides both benchmark charts and datasets: a curated collection of examples and links to existing authoring systems.

Charts and Datasets It takes a substantial amount of effort to prepare a gallery using a variety of externally-valid data and visualizations. This process includes the curation of appropriate datasets, collecting & pre-processing them, visualizing these datasets, and

documenting the results. However, it is important to respect the authorship and copyright of the source material; each research group currently needs to attain permission from the dataset owners as well as the visualization authors if a visualization design is reproduced. We envision a catalog or gallery of visualization content (which could be similar to the *Data Visualisation Catalogue* [238]), along with datasets that can be used to create this content, as well as a terms of use policy that grants permission to researchers and system developers to use the data and/or reproduce the designs. Researchers can then leverage these benchmark charts and datasets in the evaluation and comparison of visualization authoring systems.

Visualization Design Contests The development of visualization design contests may also promote the development and use of benchmark charts and datasets. Contests are a great way to produce outstanding results by engaging the members of the visualization research and practice communities, particularly novices and students. They can also complement comparative studies, which are constrained by the recruitment of study participants who may not receive sufficient training in visualization design during a brief study session. If associated with a live event such as a conference, contests can be a exciting way to motivate attendees to subsequently try out the authoring systems.

Curated Collection of Examples As discussed in Section 9.3.1, we have collected existing visualization artifacts to inform the design of several of our visualization authoring systems [70, 218, 11]. There has also been other efforts in curating collections of visualization examples, particularly in the context of data-driven storytelling [4, 239, 240]. The visualization community could benefit from such curated sets of examples, as they often include custom visualization designs that have received accolades from venues such as

the Kantar Information is Beautiful Awards[1], the Malofiej Infographic World Summit[2], and various data journalism conferences.

9.4.2 Deployment & Adoption

With the advancement of web technology, many visualization authoring systems are developed for the web environment. This makes it easy for researchers and developers to deploy the systems online and thereby reach a large audience. However, the deployment of systems and the facilitation of adoption involves a significant amount of effort that may not necessarily lead to research contributions. On the other hand, if the evaluation criteria of interest relates to how the system integrates with existing authoring workflows, or how people other than those who participated in the design of the system express themselves by using it, the deployment of the system and a study of its adoption is highly valuable.

Barriers to Studying Adoption Deploying a visualization authoring system and studying its adoption involves a substantial amount of effort to ensure that the system is usable and learnable, even when assessing its usability and learnability are not the main goals of the evaluation. In addition to engineering for usability and learnability, studying the adoption of a system entails tracking its usage.

Since it is useful to collect the content that authors produce using the system, researchers must consider ways to securely store and analyze this author-generated content. However, researchers should be aware of their ethical and legal responsibilities when storing author-generated content. The European Union's General Data Protection Regulation (GDPR), which came into effect in May 2018, has several implications for

[1] https://www.informationisbeautifulawards.com
[2] http://www.malofiejgraphics.com

researchers and particularly for those employed by industrial research labs. First, content producers using the system should opt-in to sharing their content with researchers, who must clearly state how the content will be used, such as in a research paper or online gallery. Second, content producers should be able to revoke this consent and request that researchers delete their content. Finally, while the usage tracking of any system may involve the collection of personally identifiable information (PII), visualization authoring systems are especially problematic in this regard in that it may be possible to identify individuals via the content they produce, such as when authors visualize their personal data. As a consequence, the secure storage of personally identifiable information or the de-identification of personal information are both very challenging from a practical standpoint.

Researchers may also allow visualization authors to share and discuss their content within the system. Doing so may shed light on to how people use the system to collaborate with others, which was a focus of the researchers who developed the ManyEyes [241] system. However, when including such functionality, researchers should be aware of the associated ethical and legal requirements, such as moderating both the shared content and the ensuing discussion, which entails removing objectionable content and regulating access to those who abuse the discussion platform.

Facilitating Adoption Adoption is influenced by many factors, such as the timing of the deployment, the dispersion and quality of marketing and tutorial materials, the system developers' level of influence within the target user community, and their ability to respond to technical support requests. We realize that many of these factors are difficult to control or predict, and thus a desired level of adoption is seldom guaranteed.

Despite this uncertainty, we are aware of tangible approaches to producing tutorial content for visualization authoring systems. One approach is a guided-tour tuto-

rial; ChartAccent [11] and Timeline Storyteller[218] both incorporate an Intro.js tutorial (`https://introjs.com`) that provides a step-by-step walkthrough of their interfaces. Similarly, Data Illustrator [67] provides a tutorial called "Getting Started," which mimics traditional help pages enriched with multimedia contents such as images and videos. Data Illustrator also leverages its example gallery as a way of demonstrating possible visualization designs, where each gallery image has a corresponding video documenting the visualization creation process. While such videos are great for demonstrating how to create the target visualization, it is still difficult for an individual to create one by following the video, as this process tends to involve multiple browser windows and frequent play/pause and rewind/fast-forward actions. To address this, Charticulator [12] provides a video player augmented with content segmentation and accompanying textual descriptions, a design inspired by Pluralsight (`https://pluralsight.com`), a popular self-learning platform. This approach is promising and versatile, and we see great potential in the design space of such interactive tutorials for visualization authoring systems.

Analysis of Author-Generated Content Assuming that such content is collected responsibly, a corpus of visualization content generated by people using a novel authoring system provides great value to the visualization research community. More immediately, it provides an indication of a system's expressiveness, complementing other approaches such as a gallery of content produced by the system developers. The analysis of author-generated content also provides an indication of who the authors are and what data they visualize. Such was the case with ManyEyes [58], a web-based visualization authoring system deployed online between 2007 and 2015, which attracted a wide variety of users who visualized, shared, and discussed an equally varied range of datasets [242]. Researchers may report upon a meta-analysis of the visualization design choices used by authors of the system and their coverage of the system's design space. Subject to

the consent of the authors, researchers may also include examples of author-generated visualization content in their research papers or on a supplemental gallery website.

Adoption Case Studies Individual use cases can illustrate the expressiveness and efficiency of the system, as well as how the system integrates with an author's workflow. One approach involves partnering with one or more content producers and studying their process of using the system with their own data in their own environment, perhaps over the course of multiple sessions or individual authoring projects, in the spirit of the Multi-dimensional In-depth Long-term Case study (MILC) approach [243].

Our use of 'case study' is in line with Sedlmair *et al.* [244]'s characterization of the term, distinguishing it from a 'usage scenario' envisioned and/or performed by the researchers. In other words, case studies involve a person who did not contribute to the design and development of a visualization authoring system. Some case studies involve the solicitation of particular individuals from the target user group; in the context of visualization authoring systems, these may be journalists, designers, or educators. Researchers may engage with these individuals before, during, and after their use of the visualization authoring system via interviews and potentially directly observing their usage.

Alternatively, case studies may involve people who used the system by their own volition without direct solicitation from the researchers. For instance, consider a visualization authoring system deployed online and advertised on social media; a journalist uses the system and publishes the visualization output alongside an online news article, which in turn is shared online. This presents an opportunity for researchers to reach out to this journalist and interview them about their experience using the system, potentially demonstrating their use of the tool, either in person or via a screen-sharing interview. This approach was taken by Brehmer *et al.* in their interviews with journalists following

their use of the Overview visualization system during the course of their journalistic investigation [245]; though Overview was a visual analysis tool, the same approach could be taken with a visualization authoring system.

9.5 Conclusion

We have discussed the difficulties in evaluating visualization authoring systems using traditional comparative studies, reflecting on our own experience. While comparative studies are appropriate for evaluating efficiency (*e.g.*, task time, error), controlled experiments are not necessarily appropriate for the diverse set of evaluation criteria relevant to interactive visualization authoring systems.

Collectively, we have in recent years developed and evaluated six visualization authoring systems using the collection of existing approaches for evaluating such systems: reproduction studies, free-form studies, design galleries, comparative studies, formative studies, and various combinations thereof. Given the complex and multi-faceted nature of authoring systems for expressive visualization design, we employed more than one evaluation method in these evaluations. Even though each evaluation method may have shortcomings, their combination helped us to improve our design and demonstrate the main contribution of the authoring system, complementing one another. We thus encourage other researchers to consider combining a few evaluation methods, not following our combinations exactly but devising their own combinations.

We would like to emphasize that it is important for researchers to deliberate and justify why their selection of evaluation methods are appropriate for supporting their intended research contributions, and to clearly provide their rationale in their research papers. This will help to set the right expectations among those reviewing these papers. We believe that it would be beneficial to the visualization research community if researchers

avoid conducting comparative studies of authoring systems, as these are often contrived and merely included to satisfy reviewers who expect some form of evaluation.

Finally, we see several opportunities for the future of evaluating visualization authoring systems, including the creation of benchmarks and methods for studying adoption through deployment. We hope that both researchers and reviewers will gain insights from this chapter, so that they might select or recommend more appropriate evaluation criteria and methods that better demonstrate the research contributions of a novel visualization authoring system.

Chapter 10

Discussion and Future Work

In this dissertation, we have presented multiple visualization authoring systems: iVisDesigner, ChartAccent, Charticulator, Stardust, and Idyll-MR. In this chapter, we discuss the general issues and limitations pertaining to these systems and point out future research directions.

10.1 Expressivity vs. Learnability

When designing visualization authoring systems, or authoring systems in general, there is always a tradeoff between expressivity and learnability. To make a system more expressive, we have to accommodate a wider spectrum of design options, thus the system will become more complex and harder to learn. On the other hand, to improve learnability we have to simplify the system by making assumptions on what part of the design space will more likely be used. Such assumptions have negative impacts on expressiveness.

The general-purpose visualization authoring systems we have built — iVisDesigner and Charticulator, chose to optimize for expressivity. While it is possible to create a wide variety of visualizations with them, they are harder to learn than traditional template-

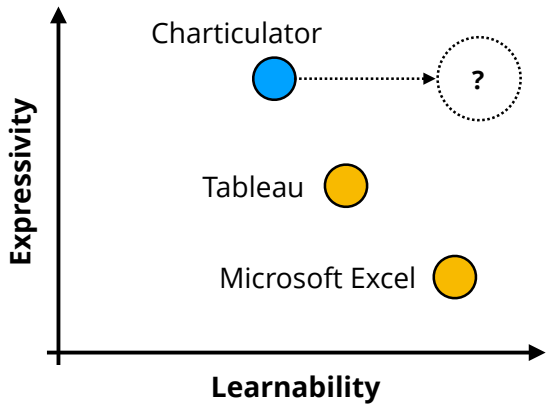


Figure 10.1: The tradeoff between expressivity and learnability. Is there a path to make an authoring tool such as Charticulator more easily learnable?

based approaches such as Microsoft Excel, or shelf-configuration approaches such as Tableau. Figure 10.1 shows an overview of visualization authoring systems in the expressivity vs. learnability space.

Visualization authoring still requires programmatic thinking One important reason why visualization authoring systems are difficult to learn is because they still require some level of programmatic thinking, albeit not programming. Except for the most simple ones, visualizations contain aspects of data mapping — a glyph corresponds to one or a group of data element. In programming languages, generating multiple glyphs from a collection of data items is usually expressed in terms of loops or map operations. iVisDesigner exposes such operations as *data enumeration*, and allows people to pick the level of enumeration from a panel; Charticulator by default maps each data item to a single glyph. The need to understand the data mapping aspect is a considerable barrier towards learning to use such systems.

Future research opportunities The tradeoffs between expressivity and learnability largely depend on the interaction modalities being used. Currently, most visualization

authoring tools use traditional WIMP-style user interfaces. However, WIMP-style user interfaces have many limitations. For example, direct manipulation works only when the target is a visible object and the manipulation gesture aligns with common expectation (*e.g.*, dragging a target should change its position). When there is no visible target (*e.g.*, a *stacking layout* that the author may want to use is not a visible object), we have to use conventional panels or menus, which can be verbose and hard to navigate. Multi-modal interactions such as pen/touch and speech can help us alleviate this problem. For example, people can apply the *packing layout* by uttering *"packing"* while pressing the plot segment or *"apply packing"* after invoking the speech input. Enabling multi-modal interactions for visualization authoring is a promising direction for future research.

I believe artificial intelligence techniques including machine learning and automated reasoning can facilitate the visualization authoring process. With new interaction modalities such as pen, multi-touch, and speech, users have much more freedom on what input they can provide to the authoring tool than with traditional WIMP-style interfaces. It is very important that the tool be able to understand the intention of the user. Figure 10.2 shows a potential workflow of future visualization authoring systems. There are many research questions. For example, how to recognize a hand-drawn sketch of a custom visualization and convert it to an actual visualization? If the sketch is provided incrementally, can the system gradually generate what the user wants? Can the system improve itself by learning from its successes and mistakes?

10.2 Reusability and Interoperability

Reusability In many scenarios, there is a need to create reusable visualization templates instead of single artifacts. For example, a sports journalist may find a particular visualization design useful for showing data from NBA matches. Being able to create

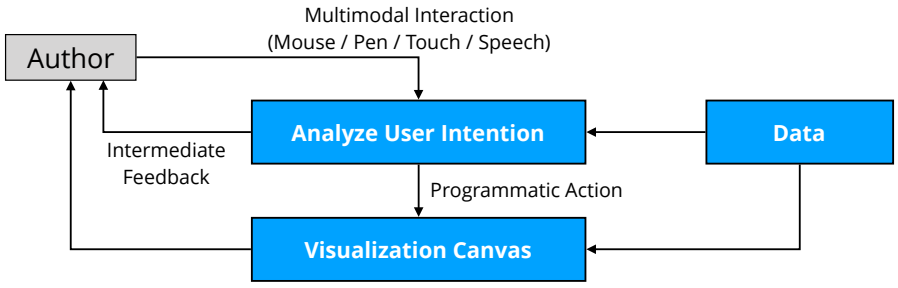


Figure 10.2: A potential workflow for future visualization authoring systems.

the visualization once and reuse it in subsequent matches can improve the efficiency of the journalist's workflow. There are multiple levels of reusability. A visualization design can be reused when data values change, but the semantics remain the same (*e.g.*, still NBA matches, but with different scores); in some cases, a visualization design may also be reused when data semantics change (*e.g.*, to soccer matches; or include additional data categories). Most existing visualization authoring tools do not support the second level of reusability. Although Charticulator supports exporting visualization designs as reusable chart templates, these templates can only be used for data with the same number and kinds of attributes. In addition, compared to Charticulator's visualization authoring capabilities, the export-to-template feature is still at a prototype stage. Providing better support for reusable visualization designs is a direction for future work.

Interoperability Visualization authoring is not an independent process. Before visualization construction, authors need to collect and prepare data; after visualization construction, the resulting artifacts are usually incorporated into the desired presentation medium, such as presentation slides, web pages, videos, or printed documents. Charticulator covers a range of export scenarios. It allows authors to export the designed visualizations into bitmap images (PNG, JPEG), vector graphics (SVG), interactive web pages (HTML), and Custom Visuals for Microsoft Power BI [179]. The static formats

(PNG, JPEG, SVG) can be used for presentation slides or printed documents; the dynamic formats (HTML and Power BI Custom Visual) can be embedded in interactive media such as web pages. While enhancing interoperability is often more of an engineering effort — implementing support for a wide variety of input/output formats — there are still future research opportunities. For example, can we design a general declarative format for the output of visualization authoring tools? Is a formalism such as Vega [162] sufficient to serve this purpose? Can we design a tool-agnostic API that serves as an intermediate format between data wrangling tools, visualization authoring tools, and presentation tools?

10.3 Dynamic, Interactive, and Animated Visualizations

Up to now, we have dealt predominantly with static visualizations, which do not animate nor respond to data change or user input.

Responses to data change While iVisDesigner and Charticulator both allow a user to replace the dataset once a visualization has been created, the resulting change of the visualization is not controllable by the user. Many well-designed visualizations use animations or annotations to highlight the newly added or changed elements. Can we design a tool to allow a user easily create visualizations that can respond to data change?

Animation Animation in general is an under-explored area for visualization authoring tools. The tools described in this dissertation, and other tools such as Data Illustrator [67] and Lyra [8] all deal with static visualizations. Animation is very important in data-driven storytelling. Well-designed animations can make a visualization look much more engaging, they can also be used to emphasis elements of interest, or point out trends and important changes in the data. What is a good workflow to create animated visualiza-

tions? What are the proper interaction design for authoring animations for visualizations? These are important directions for future research.

Interaction Card et al.'s definition of information visualization is *"the use of computer-supported, interactive, visual representations of abstract data to amplify cognition"* [1]. Despite the fact that Card emphasized the interactive aspect of information visualization and interactivity has been an important area in recent visualization specification languages [162] and libraries [7]. There is little support for interactivity in current visualization authoring tools. While iVisDesigner allows an author to configure parameter-based interactions and Charticulator enables select & highlight interactions for its HTML exports, they only cover a very small subset of the vast possibilities of interactive visualizations. To my knowledge, none of the current visualization authoring user interfaces truly support the authoring of interactive visualizations. Therefore, supporting interactive visualizations is an important direction for future research.

10.4 Immersive Visualization Authoring User Interfaces

Visualization authoring user interfaces for immersive environments is still under-explored. None of current systems offers the same level of expressivity as desktop-based visualization authoring tools. It is challenging to design immersive user interfaces and there are still many open questions. Visualization authoring brings additional challenges for immersive user interface design. Do techniques used in iVisDesigner and Charticulator (*e.g.*, drop-zones, configuration panels) work in immersive environments? What are the required modifications? Perhaps more importantly, we still need to better understand the design space of immersive visualizations. Augmented reality presents the opportunity for connecting visualizations to the physical environment; immersive envi-

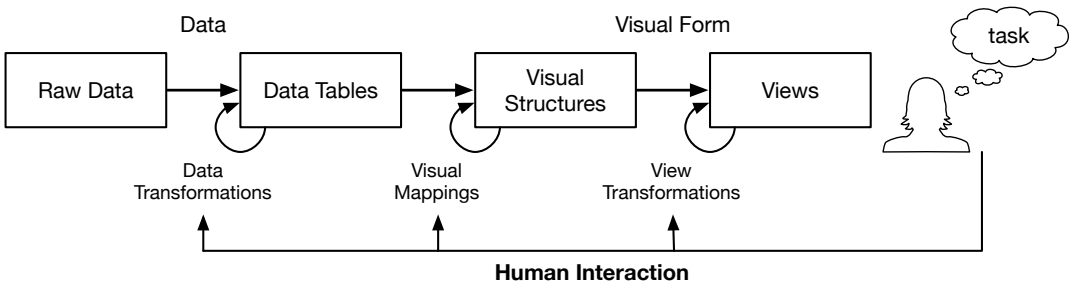


Figure 10.3: Card [1]'s visualization reference model

ronments also provide a vast amount of available space for displaying information. What forms of visualizations are useful in immersive environments? Do 3D visualizations have an advantage over 2D ones given the better navigation capabilities in immersive environments?

10.5 Beyond Visualization Authoring

In this dissertation, we have scoped "visualization authoring" as the process of creating a given visualization design for a given dataset. However, in the real world, an author rarely has a predefined visualization design with a given dataset already in mind before getting to use a tool to create the visualization. The process to visualize data is often highly organic and iterative (see Figure 10.3).

Data import and processing Most current visualization authoring tools require the input data to be prepared in the right format. This is assuming that the author (1) can figure out what is the right data format; and (2) is able to gather and transform the data into the right format. The former requires knowledge of how the visualization authoring tool works; the latter requires expertise in data wrangling tools or programming. When we give our tools (*e.g.*, Charticulator) to people to visualize their own data, we often observe them struggling more with preparing the data than with using the tool to create a

Wide Format

Month	Seattle	New York	Anchorage
January	45	39	23
February	48	42	27
March	52	50	34
April	58	60	44
May	64	71	56
June	69	79	63
…	…	…	…

12 rows

Long Format

Month	City	Temperature
January	Seattle	45
January	New York	39
January	Anchorage	23
February	Seattle	48
February	New York	42
February	Anchorage	27
…	…	…

36 rows

Figure 10.4: An example data with wide format (left) and long format (right). The data consists of monthly temperature values of three cities. In the wide format we have one column for each city, one row for each month; in the long format, we have a `City` column and a `Temperature` column, with 12×3 rows.

visualization. One issue that occurs very often is the choice between *wide format* and *long format*. For example, in Figure 10.4, we have a dataset consisting of monthly temperature values for three cities. The same data can be represented in either a wide format or a long format. When imported into a tool like Charticulator or Data Illustrator [67], the two formats are interpreted differently even though they represent essentially the same data. Because they are interpreted differently, the sequence of interactions to create a chart is different. Is it possible to resolve this discrepancy in visualization authoring tools? In the future, we should investigate how to better integrate data wrangling and preprocessing functionality into visualization authoring tools.

Design tool vs. authoring tool One fundamental assumption of current visualization authoring tools is that the author already have a design in mind prior to using such tools. The design could be done by sketching on paper, or inspired by some other design the author saw elsewhere. This assumption also implies that the author is already familiar with the dataset (as a result of prior data exploration). Such assumptions may not hold true in reality. If authors start without prior data exploration, or without a specific design

in mind, they will face a blank canvas with little help to get started. On the other hand, it is also unrealistic to assume a linear design process. In practice, the author may iteratively go through the data exploration, design, and authoring processes. We anticipate that future visualization authoring systems could relax these assumptions to facilitate such an iterative design process. For example, how to enable authors to quickly explore and keep track of different design ideas? Can we facilitate the sharing of design ideas? To alleviate the cold start problem, can we design a creativity support system that can automatically suggest design options? Can the system help authors to avoid common design pitfalls?

10.6 Deployment of Visualization Authoring Tools

We believe that the ultimate test for visualization authoring tools is their deployment and adoption. Most of the tools described in this dissertation have been released publicly and with open source licenses:

1. iVisDesigner: `https://github.com/donghaoren/ivisdesigner/`
2. ChartAccent: `https://chartaccent.github.io/`
3. Charticulator: `https://charticulator.com/`
4. Stardust: `https://stardustjs.github.io/`

Given their public and open source releases, it is possible for future researchers to reproduce our results and conduct further evaluations.

As we have discussed in Chapter 9, it is difficult to scientifically analyze the results of such deployment, because the actual adoption after a public release largely depends on marketing and advertising efforts. As a preliminary example result, as of March 2019,

Charticulator had around 100 unique users per day with 12k page views per month. Its GitHub repository received 202 stars and has been forked 33 times. We have also received various bug reports and feature requests. We hope that in the future it can be turned into a real product and continue to grow.

Chapter 11

Conclusions

In this dissertation, we have investigated the problem of user interface design for visualization authoring, with a primary focus on creating visualizations for data-driven storytelling. We first developed and evaluated tools for visualization authoring in traditional user interfaces concerning three aspects: data mapping, layout, and annotation. Then, we extended the tools to support immersive environments.

In Part I, we have presented iVisDesigner, ChartAccent, and Charticulator. We started out by designing iVisDesigner, a new expressive interactive information visualization construction toolkit. It covers a broader spectrum of possible visualization designs than previous interactive (non-programming) toolkits. ChartAccent, on the other hand, supports authoring data-driven annotations through a well-designed user interface informed by an analysis of current annotation practices. There are two limitations of iVisDesigner: (1) The learning curve is high. When users have to learn a whole set of new concepts, such as axes, references, and components, it will inevitably take some time for them to embrace the possibilities and fully utilize their potential for creative designs. (2) The data-flow framework cannot well encode the relationships between graphical elements. These drawbacks have been addressed in Charticulator by (1) clearly conveying the de-

sign concepts in the user interface and providing much more versatile interactions than offered by control panels and mouse clicking; and (2) introducing the concept of composable layouts and incorporating a constraint solver to deal with inter-graphical-element layout relationships.

We have evaluated our visualization authoring approaches regarding to two crucial aspects: (1) the expressiveness of the systems are evaluated through demonstrated usage scenarios; (2) the usability and learnability of the tools are evaluated through usability evaluations or workshop evaluations. However, the usability evaluations are tied to the particular tools and application scenarios being evaluated. Therefore the findings are not readily generalizable. Furthermore, in a controlled experiment, it is not easily possible to measure long-term learning, usage, and adoption. Due to these reasons, we believe that the ultimate evaluation of these tools is through releasing them to the public, and continuously measuring user adoption and collecting feedback. In this way, we will also empower a broader community to design richer visualizations for data-driven storytelling.

In Part II, we have presented tools to create visualizations for virtual and augmented reality environments. First, we have shown a simple visualization creation system targeting projection-based spherical virtual environments. The system was used to design and conduct a user study that evaluates the effect of field of view in wide-field-of-view augmented reality [9]. The simple visualization authoring system only supports 2D charts in spherical space and basic linking between them. To ease the development of complex visualizations in virtual and augmented reality, we have built and evaluated a new visualization library called Stardust. It provides a cross-platform and familiar API to create scalable visualizations through GPU acceleration. Finally, we have presented Idyll-MR, a declarative language for creating immersive data-driven stories.

In Part III, we have presented discussions on the evaluation of interactive visualization

authoring systems, as well as on general issues and limitations pertaining to these systems. We hope that the systems presented in this dissertation can help practitioners construct more expressive visualizations for data-driven storytelling and inspire future research on interactive visualization authoring tools.

Bibliography

[1] S. K. Card, J. D. Mackinlay, and B. Schneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[2] J. J. Thomas and K. A. Cook, *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005.

[3] B. Lee, N. Henry Riche, P. Isenberg, and S. Carpendale, *More than telling a story: Transforming data into visually shared stories*, *IEEE Computer Graphics and Applications* **35** (2015), no. 5 84–90.

[4] E. Segel and J. Heer, *Narrative visualization: Telling stories with data*, *IEEE Transactions on Visualization and Computer Graphics* **16** (2010), no. 6 1139–1148.

[5] B. Victor, "Drawing dynamic visualizations." `http://vimeo.com/66085662`, 2013.

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

[7] M. Bostock, V. Ogievetsky, and J. Heer, *D3: Data-driven documents*, *IEEE Transactions on Visualization and Computer Graphics* **17** (2011), no. 12 2301–2309.

[8] A. Satyanarayan and J. Heer, *Lyra: An interactive visualization design environment*, *Computer Graphics Forum* **33** (2014), no. 3.

[9] D. Ren, T. Goldschwendt, Y. Chang, and T. Höllerer, *Evaluating wide-field-of-view augmented reality with mixed reality simulation*, in *Proceedings of IEEE Virtual Reality (VR)*, pp. 93–102, 2016.

[10] D. Ren, T. Höllerer, and X. Yuan, *iVisDesigner: Expressive interactive design of information visualizations*, *IEEE Transactions on Visualization and Computer Graphics* **20** (2014), no. 12 2092–2101.

[11] D. Ren, M. Brehmer, B. Lee, T. Höllerer, and E. K. Choe, *ChartAccent: Annotation for data-driven storytelling*, in *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, pp. 230–239, 2017.

[12] D. Ren, B. Lee, and M. Brehmer, *Charticulator: Interactive construction of bespoke chart layouts*, *IEEE Transactions on Visualization and Computer Graphics* **25** (2019), no. 1.

[13] D. Ren, B. Lee, and T. Höllerer, *Stardust: Accessible and transparent GPU support for information visualization rendering*, *Computer Graphics Forum* (2017).

[14] D. Ren, B. Lee, M. Brehmer, and N. Henry Riche, *Reflecting on the evaluation of visualization authoring systems : Position paper*, in *Proceedings of IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*, pp. 86–92, 2018.

[15] J. Bertin, *Sémiologie Graphique*. Editions Gauthier-Villars, Paris, France, 1967.

[16] J. Bertin, *Semiology of graphics*. W. Berg, transl., University of Wisconsin Press, Madison, Wisconsin, 1983.

[17] J. Mackinlay, *Automating the design of graphical presentations of relational information*, *ACM Transactions Graphics* **5** (1986), no. 2 110–141.

[18] B. Shneiderman, *The eyes have it: A task by data type taxonomy for information visualizations*, in *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pp. 336–343, 1996.

[19] S. K. Card and J. Mackinlay, *The structure of the information visualization design space*, in *Proceedings of the IEEE Symposium on Information Visualization*, pp. 92–99, 1997.

[20] R. Kosara and J. Mackinlay, *Storytelling: The next step for visualization*, *IEEE Computer* **46** (2013), no. 5 44–50.

[21] C. D. Stolper, B. Lee, N. Henry Riche, and J. Stasko, *Emerging and recurring data-driven storytelling techniques: Analysis of a curated collection of recent stories*, Tech. Rep. MSR-TR-2016-14, Microsoft Research, 2016.

[22] E. K. Choe, B. Lee, and M. c. schraefel, *Characterizing visualization insights from quantified selfers' personal data presentations*, *IEEE Computer Graphics and Applications* **35** (2015), no. 4 28–37.

[23] J. Hullman and N. Diakopoulos, *Visualization rhetoric: Framing effects in narrative visualization*, *IEEE Transactions on Visualization and Computer Graphics* **17** (2011), no. 12 2231–2240.

[24] J. Boy, F. Detienne, and J.-D. Fekete, *Storytelling in information visualizations: Does it engage users to explore data?*, in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, (New York, NY, USA), pp. 1449–1458, ACM, 2015.

[25] K. Ma, I. Liao, J. Frazier, H. Hauser, and H. Kostis, *Scientific storytelling using visualization*, *IEEE Computer Graphics and Applications* **32** (2012), no. 1 12–19.

[26] C. Reas and B. Fry, *Processing: Programming for the media arts*, *AI & Society* **20** (2006), no. 4 526–538.

[27] J.-D. Fekete, *The InfoVis toolkit*, in *Proceedings of the IEEE Symposium on Information Visualization*, pp. 167–174, IEEE, 2004.

[28] C. Weaver, *Building highly-coordinated visualizations in improvise*, in *Proceedings of the IEEE Symposium on Information Visualization*, pp. 159–166, 2004.

[29] E. H.-H. Chi, P. Barry, J. Riedl, and J. Konstan, *A spreadsheet approach to information visualization*, in *Proceedings of the IEEE Symposium on Information Visualization*, pp. 17–24, 1997.

[30] A. G. Forbes, T. Höllerer, and G. Legrady, *"behaviorism": a framework for dynamic data visualization*, *Visualization and Computer Graphics, IEEE Transactions on* **16** (2010), no. 6 1164–1171.

[31] J. Heer, S. K. Card, and J. A. Landay, *Prefuse: A toolkit for interactive information visualization*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pp. 421–430, ACM, 2005.

[32] "Flare." `http://flare.prefuse.org/`, 2014.

[33] L. Wilkinson, *The Grammar of Graphics*. Springer-Verlag, 1999.

[34] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer Publishing Company, Incorporated, 2nd ed., 2009.

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

[36] A. Satyanarayan, K. Wongsuphasawat, and J. Heer, *Declarative interaction design for data visualization*, in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pp. 669–678, ACM, 2014.

[37] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, *Vega-Lite: A grammar of interactive graphics*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 341–350.

[38] J. Kruger and R. Westermann, *Acceleration techniques for GPU-based volume rendering*, in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, (Washington, DC, USA), pp. 38–38, IEEE Computer Society, 2003.

[39] M. Falk, S. Grottel, M. Krone, and G. Reina, *Interactive GPU-based Visualization of Large Dynamic Particle Data*, vol. 4(3) of *Synthesis Lectures on Visualization*. Morgan & Claypool Publishers, San Rafael, CA, 2016.

[40] W. J. Schroeder, B. Lorensen, and K. Martin, *The visualization toolkit.* Kitware, 2004.

[41] A. Sarikaya and M. Gleicher, *Using WebGL as an interactive visualization medium: Our experience developing SplatterJs*, in *Proceedings of the Data Systems for Interactive Analysis Workshop* (D. F. J. H. Remco Chang Carlos Scheidegger, ed.), IEEE, 2015. DSIA '15.

[42] K. Andrews and B. Wright, *FluidDiagrams: Web-based information visualisation using JavaScript and WebGL*, in *EuroVis — Short Papers* (N. Elmqvist, M. Hlawitschka, and J. Kennedy, eds.), The Eurographics Association, 2014.

[43] G. G. Robertson, J. D. Mackinlay, and S. K. Card, *Cone Trees: Animated 3D visualizations of hierarchical information*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pp. 189–194, ACM, 1991.

[44] S. Drucker and R. Fernandez, *A unifying framework for animated and interactive unit visualizations*, tech. rep., Microsoft Research, 2015.

[45] Z. Liu, B. Jiang, and J. Heer, *imMens: Real-time visual querying of big data*, *Computer Graphics Forum* **32** (2013), no. 3 421–430.

[46] D. Holten and J. J. van Wijk, *Force-directed edge bundling for graph visualization*, in *Proceedings of the 11th Eurographics / IEEE – VGTC Conference on Visualization*, EuroVis '09, pp. 983–998, The Eurographs Association & John Wiley & Sons, Ltd., 2009.

[47] M. McCool, S. Du Toit, T. Popa, B. Chan, and K. Moule, *Shader algebra*, *ACM Transactions on Graphics* **23** (2004), no. 3 787–795.

[48] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, *Brook for GPUs: stream computing on graphics hardware*, *ACM Transactions on Graphics (TOG)* **23** (2004), no. 3 777–786.

[49] L. Grammel, C. Bennett, M. Tory, and M.-A. Storey, *A survey of visualization construction user interfaces*, in *Short Paper Proceedings of EuroVis*, 2013.

[50] P. E. Haeberli, *Conman: a visual programming language for interactive graphics*, in *ACM SIGGRAPH Computer Graphics*, vol. 22, pp. 103–111, ACM, 1988.

[51] C. Upson, T. A. Faulhaber Jr, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. Van Dam, *The application visualization system: A computational environment for scientific visualization*, *Computer Graphics and Applications, IEEE* **9** (1989), no. 4 30–42.

[52] D. Foulser, *IRIS Explorer: A framework for investigation*, *ACM SIGGRAPH Computer Graphics* **29** (1995), no. 2 13–16.

[53] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo, *VisTrails: enabling interactive multiple-view visualizations*, in *Proceedings of 2005 IEEE Visualization*, pp. 135–142, 2005.

[54] P. T. Inc., *Collaborative data science*, 2015.

[55] "ChartBlocks: The online chart building tool." `https://www.chartblocks.com/en/`, accessed Nov. 20th, 2017.

[56] DensityDesign, "RAWGraphs: The missing link between spreadsheets and data visualization." `https://rawgraphs.io/`, accessed Nov. 20th, 2017.

[57] "Quadrigram." `http://www.quadrigram.com/`, accessed Nov. 20th, 2017.

[58] F. B. Viégas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon, *ManyEyes: A site for visualization at internet scale*, *IEEE Transactions on Visualization and Computer Graphics* **13** (2007), no. 6 1121–1128.

[59] J. Heer, F. B. Viégas, and M. Wattenberg, *Voyagers and voyeurs: Supporting asynchronous collaborative information visualization*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1029–1038, 2007.

[60] W. Willett, J. Heer, J. Hellerstein, and M. Agrawala, *CommentSpace: Structured support for collaborative visual analysis*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 3131–3140, 2011.

[61] Tableau Software. `http://tableau.com`.

[62] J. H. T. Claessen and J. J. van Wijk, *Flexible linked axes for multivariate data visualization*, *IEEE Transactions on Visualization and Computer Graphics* **17** (2011), no. 12 2310–2316.

[63] B. Lee, R. Kazi, and G. Smith, *Sketchstory: Telling more engaging stories with data through freeform sketching*, *Visualization and Computer Graphics, IEEE Transactions on* **19** (2013), no. 12 2416–2425.

[64] J. Walny, B. Lee, P. Johns, N. Riche, and S. Carpendale, *Understanding pen and touch interaction for data exploration on interactive whiteboards*, *Visualization and Computer Graphics, IEEE Transactions on* **18** (2012), no. 12 2779–2788.

[65] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein, *Interactive graphic design using automatic presentation knowledge*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 112–117, ACM, 1994.

[66] UW Interactive Data Lab, "Vega: A visualization grammar." `https://github.com/vega/vega`.

[67] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko, *Data Illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 123:1–123:13, 2018.

[68] H. Mei, W. Chen, Y. Ma, H. Guan, and W. Hu, *VisComposer: A visual programmable composition environment for information visualization*, *Visual Informatics* **2** (2018), no. 1 71–81.

[69] A. Satyanarayan and J. Heer, *Authoring narrative visualizations with Ellipsis*, *Computer Graphics Forum* **33** (2014), no. 3 361–370.

[70] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani, *Authoring data-driven videos with dataclips*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 501–510.

[71] M. Brehmer, B. Lee, B. Bach, N. Henry Riche, and T. Munzner, *Timelines revisited: A design space and considerations for expressive storytelling*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 9 2151–2164.

[72] M. Elias, *Enhancing User Interaction with Business Intelligence Dashboards.* PhD thesis, École Centrale Paris, 2012.

[73] J. Heer and B. Shneiderman, *Interactive dynamics for visual analysis: A taxonomy of tools that support the fluent and flexible use of visualizations*, *Comm. ACM* (2012) 1–26.

[74] J. Heer, M. Agrawala, and W. Willett, *Generalized selection via interactive query relaxation*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 959–968, 2008.

[75] Y. Chen, S. Barlowe, and J. Yang, *Click2Annotate: Automated insight externalization with rich semantics*, in *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pp. 155–162, 2010.

[76] Y. Chen, J. Yang, S. Barlowe, and D. H. Jeong, *Touch2Annotate: Generating better annotations with less human effort on multi-touch interfaces*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 3703–3708, 2010.

[77] E. Kandogan, *Just-in-time annotation of clusters, outliers, and trends in point-based data visualizations*, in *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 73–82, 2012.

[78] C. Bryan, K.-L. Ma, and J. Woodring, *Temporal summary images: An approach to narrative visualization via interactive annotation generation and placement*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 511–520.

[79] L. H. Owen, *At the Malofiej Infographics World Summit, "The best form of storytelling is often static"*, *NiemenLab* (2016). https://www.niemanlab.org/2016/03/at-the-malofiej-infographics-world-summit-the-best-form-of-storytelling-is-often-static/.

[80] L. C. Rost, *What I learned recreating one chart using 24 tools*, *Source* (2016). https://source.opennews.org/articles/what-i-learned-recreating-one-chart-using-24-tools/.

[81] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer, *Reflections on how designers design with data*, in *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI)*, pp. 17–24, 2014.

[82] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer, *Iterating between tools to create and edit visualizations*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 481–490.

[83] G. Aisch, "Seven features you'll want in your next charting tool." Presented at NICAR, 2015. https://www.vis4.net/blog/2015/03/seven-features-youll-wantin-your-next-charting-tool/.

[84] A. Pearce, *swoopyDrag.js: Artisinal label placement for d3 graphics*, 2016. http://1wheel.github.io/swoopy-drag/.

[85] K. Wongsuphasawat, *labella.js*, 2015. http://twitter.github.io/labella.js/.

[86] J. Moloney, A. Borning, and B. Freeman-Benson, *Constraint technology for user-interface construction in ThingLab II*, *ACM SIGPLAN Notices* **24** (1989), no. 10.

[87] J. Fogarty and S. E. Hudson, *GADGET: A toolkit for optimization-based approaches to interface and display generation*, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 125–134, 2003.

[88] "Android developers constraint library." https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html.

[89] K. Ryall, J. Marks, and S. Shieber, *An interactive constraint-based system for drawing graphs*, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 97–104, 1997.

[90] I. F. Cruz and P. S. Leveille, *Implementation of a constraint-based visualization system*, in *Proceedings of the IEEE Symposium on Visual Languages*, pp. 13–20, 2000.

[91] S. Takahashi, S. Matsuoka, K. Miyashita, H. Hosobe, and T. Kamada, *A constraint-based approach for visualization and animation*, *Constraints* **3** (1998), no. 1 61–86.

[92] D. A. Bowman and R. P. McMahan, *Virtual reality: How much immersion is enough?*, *Computer* **40** (2007), no. 7 36–43.

[93] K. W. Arthur, *Effects of field of view on performance with head-mounted displays*. PhD thesis, 2000.

[94] J. M. Covelli, J. P. Rolland, M. Proctor, J. P. Kincaid, and P. Hancock, *Field of view effects on pilot performance in flight*, *The International Journal of Aviation Psychology* **20** (2010), no. 2 197–219.

[95] J. A. Jones, J. E. Swan, II, G. Singh, and S. R. Ellis, *Peripheral visual information and its effect on distance judgments in virtual and augmented environments*, in *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*, APGV '11, (New York, NY, USA), pp. 29–36, ACM, 2011.

[96] E. D. Ragan, D. A. Bowman, R. Kopper, C. Stinson, S. Scerbo, and R. P. McMahan, *Effects of field of view and visual complexity on virtual reality training effectiveness for a visual scanning task*, *IEEE Transactions on Visualization and Computer Graphics* **21** (2015), no. 7 794–807.

[97] J. M. Knapp and J. M. Loomis, *Limited field of view of head-mounted displays is not the cause of distance underestimation in virtual environments*, *Presence: Teleoper. Virtual Environ.* **13** (2004), no. 5 572–577.

[98] R. Ball and C. North, *The effects of peripheral vision and physical navigation on large scale visualization*, in *Proceedings of Graphics Interface 2008*, GI '08, (Toronto, Ont., Canada, Canada), pp. 9–16, Canadian Information Processing Society, 2008.

[99] J. Ventura, M. Jang, T. Crain, T. Höllerer, and D. Bowman, *Evaluating the effects of tracker reliability and field of view on a target following task in augmented reality*, in *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, pp. 151–154, 2009.

[100] T. J. Buker, D. A. Vincenzi, and J. E. Deaton, *The effect of apparent latency on simulator sickness while using a see-through helmet-mounted display: reducing apparent latency with predictive compensation.*, *Human Factors* **54** (2012), no. 2 235–49.

[101] N. Kishishita, J. Orlosky, T. Mashita, K. Kiyokawa, and H. Takemura, *Poster: Investigation on the peripheral visual field for information display with real and virtual wide field-of-view see-through hmds*, in *Proceedings of the 2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pp. 143–144, 2013.

[102] D. Van Nguyen, T. Mashita, K. Kiyokawa, and H. Takemura, *Subjective image quality assessment of a wide-view head mounted projective display with a semi-transparent retro-reflective screen*, in *Proceedings of the 21st International Conference on Artificial Reality and Telexistence (ICAT)*, 2011.

[103] N. Kishishita, K. Kiyokawa, J. Orlosky, T. Mashita, H. Takemura, and E. Kruijff, *Analysing the effects of a wide field of view augmented reality display on search performance in divided attention tasks*, in *Proceedings of the 2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 177–186, 2014.

[104] M. Livingston, *Evaluating human factors in augmented reality systems*, *IEEE Computer Graphics and Applications* **25** (2005), no. 6 6–9.

[105] T. Olsson, T. Kärkkäinen, E. Lagerstam, and L. Ventä-Olkkonen, *User evaluation of mobile augmented reality scenarios*, *Journal of Ambient Intelligence and Smart Environments* **4** (2012), no. 1 29–47.

[106] J. Swan and J. Gabbard, *Survey of user-based experimentation in augmented reality*, in *Proceedings of the 1st HCI International Conference on Virtual Reaity*, 2005.

[107] A. Dünser, R. Grasset, and M. Billinghurst, *A survey of evaluation techniques used in augmented reality studies*, tech. rep., University of Canterbury, 2008.

[108] N. Kostaras and M. Xenos, *Usability evaluation of augmented reality systems*, *Intelligent Decision Technologies* **6** (2012), no. 2 139–149.

[109] K. Kiyokawa, *A wide field-of-view head mounted projective display using hyperbolic half-silvered mirrors*, in *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–4, IEEE, 2007.

[110] A. Maimone, D. Lanman, K. Rathinavel, K. Keller, D. Luebke, and H. Fuchs, *Pinlight displays*, *ACM Transactions on Graphics* **33** (2014), no. 4 1–11.

[111] "Innovega Inc. iOptik platform: A new architecture." `http://innovega-inc.com/new-architecture.php`, accessed September 2015.

[112] T. Oskiper, M. Sizintsev, V. Branzoi, S. Samarasekera, and R. Kumar, *Augmented reality binoculars*, *IEEE Transactions on Visualization and Computer Graphics* **21** (2015), no. 5 611–623.

[113] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister, *Data-driven guides: Supporting expressive design for information graphics*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 491–500.

[114] Y. Wang, H. Zhang, H. Huang, X. Chen, Q. Yin, Z. Hou, D. Zhang, Q. Luo, and H. Qu, *InfoNice: Easy creation of information graphics*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 335:1–335:12, 2018.

[115] M. Cavazza, J.-L. Lugrin, D. Pizzi, and F. Charles, *Madame bovary on the holodeck: immersive interactive storytelling*, in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 651–660, ACM, 2007.

[116] R. Sicat, J. Li, J. Choi, M. Cordeil, W.-K. Jeong, B. Bach, and H. Pfister, *Dxr: A toolkit for building immersive data visualizations*, *IEEE Transactions on Visualization and Computer Graphics* **25** (2019), no. 1 715–725.

[117] M. Cordeil, A. Cunningham, B. Bach, C. Hurter, B. H. Thomas, K. Marriott, and T. Dwyer, *IATK: An immersive analytics toolkit*, in *Proceedings of the IEEE Virtual Reality Conference (VR '19)*, 2019.

[118] P. Isenberg, B. Lee, H. Qu, and M. Cordeil, *Immersive visual data stories*, in *Immersive Analytics*. Springer, 2018.

[119] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, *et. al.*, *Immersive and collaborative data visualization using virtual reality platforms*, in *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 609–614, IEEE, 2014.

[120] S. Gebhardt, T. Petersen-Krau, S. Pick, D. Rausch, C. Nowke, T. Knott, P. Schmitz, D. Zielasko, B. Hentschel, and T. W. Kuhlen, *Vista widgets: a framework for designing 3D user interfaces from reusable interaction building blocks*, in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pp. 251–260, ACM, 2016.

[121] M. Cordeil, A. Cunningham, T. Dwyer, B. H. Thomas, and K. Marriott, *ImAxes: Immersive axes as embodied affordances for interactive multivariate data visualisation*, in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, (New York, NY, USA), pp. 71–83, ACM, 2017.

[122] K.-D. Le, M. Fjeld, A. Alavi, and A. Kunz, *Immersive environment for distributed creative collaboration*, in *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, p. 16, ACM, 2017.

[123] M. Cavallo, M. Dolakia, M. Havlena, K. Ocheltree, and M. Podlaseck, *Dataspace: A reconfigurable hybrid reality environment for collaborative information analysis*, in *Proceedings of the IEEE Virtual Reality Conference (VR '19)*, 2019.

[124] M. Cordeil, T. Dwyer, K. Klein, B. Laha, K. Marriott, and B. H. Thomas, *Immersive collaborative analysis of network connectivity: CAVE-style or head-mounted display?*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 441–450.

[125] B. Bach, R. Sicat, J. Beyer, M. Cordeil, and H. Pfister, *The Hologram in my hand: How effective is interactive exploration of 3D visualizations in immersive tangible augmented reality?*, *IEEE Transactions on Visualization and Computer Graphics* **24** (2018), no. 1 457–467.

[126] Y. Yang, T. Dwyer, B. Jenny, K. Marriott, M. Cordeil, and H. Chen, *Origin-destination flow maps in immersive environments*, *IEEE Transactions on Visualization and Computer Graphics* **25** (2019), no. 1 693–703.

[127] C. Andrews, A. Endert, B. Yost, and C. North, *Information visualization on large, high-resolution displays: Issues, challenges, and opportunities*, *Information Visualization* **10** (2011), no. 4 341–355.

[128] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, *The CAVE: Audio visual experience automatic virtual environment*, *Communications of the ACM* **35** (1992), no. 6 64–72.

[129] W. R. Sherman, D. Coming, and S. Su, *FreeVR: Honoring the past, looking to the future*, in *Proceedings of SPIE, The Engineering Reality of Virtual Reality*, vol. 8649, 2013.

[130] D. Burns and R. Osfield, *Tutorial: Open scene graph*, in *Proceedings of IEEE Virtual Reality*, pp. 265–265, IEEE, 2004.

[131] "TechViz." `http://www.techviz.net/`, accessed Dec. 3, 2016.

[132] "Unreal Engine." `https://www.unrealengine.com/`, accessed Dec. 3, 2016.

[133] "Unity game engine." `https://unity3d.com/`, accessed Dec. 3, 2016.

[134] H. Le, A. Joshi, and M. Betke, "b3.js: A library for interactive web data visualizations in virtual reality." `https://github.com/huyle333/b3`, accessed Dec. 3, 2016.

[135] S. Peleg, M. Ben-Ezra, and Y. Pritch, *Omnistereo: Panoramic stereo imaging*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23** (2001), no. 3 279–290.

[136] J. Kuchera-Morin, M. Wright, G. Wakefield, C. Roberts, D. Adderton, B. Sajadi, T. Höllerer, and A. Majumder, *Immersive full-surround multi-user system design*, *Computers & Graphics* **40** (2014), no. 0 10–21.

[137] H. Guo, N. Mao, and X. Yuan, *WYSIWYG (What You See is What You Get) volume visualization*, *Visualization and Computer Graphics, IEEE Transactions on* **17** (2011), no. 12 2106–2114.

[138] B. A. Myers, J. Goldstein, and M. A. Goldberg, *Creating charts by demonstration*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, (New York, NY, USA), pp. 106–111, ACM, 1994.

[139] X. Yuan, P. Guo, H. Xiao, H. Zhou, and H. Qu, *Scattering points in parallel coordinates*, *IEEE Transactions on Visualization and Computer Graphics* **15** (2009), no. 6 1001–1008.

[140] H.-J. Schulz, Z. Akbar, and F. Maurer, *A generative layout approach for rooted tree drawings*, in *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, pp. 225–232, 2013.

[141] CMU StatLib, "Car data." `http://lib.stat.cmu.edu/datasets/cars.data`.

[142] T. M. J. Fruchterman and E. M. Reingold, *Graph drawing by force-directed placement*, *Software-Practice and Experience* **21** (1991), no. 11 1129–1164.

[143] D. Ren, X. Zhang, Z. Wang, J. Li, and X. Yuan, *WeiboEvents: A crowd sourcing weibo visual analytic system*, in *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, pp. 330–334, 2014.

[144] T. H. Fay, *The butterfly curve*, *The American Mathematical Monthly* **96** (1989), no. 5 pp. 442–443.

[145] A. Kirk, *Data Visualization: A Successful Design Process*. Packt, 2012.

[146] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister, *What makes a visualization memorable?*, *IEEE Transactions on Visualization and Computer Graphics* **19** (2013), no. 12 2306–2315.

[147] D. Gotz and M. X. Zhou, *Characterizing users' visual analytic activity for insight provenance*, in *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pp. 123–130, 2008.

[148] R. E. Roth, *An empirically-derived taxonomy of interaction primitives for interactive cartography and geovisualization*, *IEEE Transactions on Visualization and Computer Graphics* **19** (2013), no. 12 2356–2365.

[149] M. A. Borkin, Z. Bylinskii, N. W. Kim, C. M. Bainbridge, C. S. Yeh, D. Borkin, H. Pfister, and A. Oliva, *Beyond memorability: Visualization recognition and recall*, *IEEE Transactions on Visualization and Computer Graphics* **22** (2016), no. 1 519–528.

[150] C. C. Marshall, *Annotation: From paper books to the digital library*, in *Proceedings of the ACM International Conference on Digital Libraries*, pp. 131–140, 1997.

[151] E. Tufte, *Layering and separation*, in *Envisioning Information*, pp. 52–65. Graphics Press, 1990.

[152] T. Grossman and R. Balakrishnan, *The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 281–290, 2005.

[153] C. Collins, G. Penn, and S. Carpendale, *Bubble sets: Revealing set relations with isocontours over existing visualizations*, *IEEE Transactions on Visualization and Computer Graphics* **15** (2009), no. 6 1009–1016.

[154] FiveThirtyEight, "Data and code behind the stories and interactives at FiveThirtyEight." GitHub repository. `https://github.com/fivethirtyeight/data`.

[155] Gapminder Foundation. `http://gapminder.org`.

[156] M. Bostock, *Force-directed graph*, 2016. `https://bl.ocks.org/mbostock/4062045`.

[157] "World Climate." `http://worldclimate.com`.

[158] "@BeijingAir." `https://twitter.com/BeijingAir`.

[159] M. Lichman, *UCI machine learning repository*, 2013. `http://archive.ics.uci.edu/ml`.

[160] A. Azzalini and A. W. Bowman, *A look at some data on the Old Faithful Geyser*, *J. Royal Statistical Society. Series C (Applied Statistics)* **39** (1990), no. 3 357–365.

[161] J. Harper and M. Agrawala, *Deconstructing and restyling D3 visualizations*, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 253–262, 2014.

[162] 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** (2016), no. 1 659–668.

[163] G. G. Méndez, M. A. Nacenta, and S. Vandenheste, *iVoLVER: Interactive visual language for visualization extraction and reconstruction*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 4073–4085, 2016.

[164] T. Herzog, World Greenhouse Gas Emissions: 2005, World Resources Institute, http://wri.org/resources/charts-graphs/world-greenhouse-gas-emissions-2005.

[165] "Project Tycho." University of Pittsburgh, www.tycho.pitt.edu.

[166] Millennium Indicators, United Nations Statistics Division. http://mdgs.un.org/unsd/mdg/SeriesDetail.aspx?srid=749.

[167] National Centers for Environmental Information. https://www.ncdc.noaa.gov/cdo-web/search?datasetid=GHCND.

[168] D. Knuth. The Stanford GraphBase: A Platform for Combinatorial Computing. https://www-cs-faculty.stanford.edu/~knuth/sgb.html.

[169] "Caltrain's schedule." http://caltrain.com/schedules/weekdaytimetable.html.

[170] A. Cairo, *The Functional Art: An Introduction to Information Graphics and Visualization.* New Riders, 2012.

[171] F. Nightingale and W. Farr and A. Smith. *A contribution to the sanitary history of the British army during the late war with Russia.* John W. Parker and Son, 1859.

[172] T. Kim. Best Bookshelf. http://tany.kim/best-bookshelf.

[173] "StatCounter: Mobile operating system market share worldwide." http://gs.statcounter.com/os-market-share/mobile/worldwide.

[174] T. DeBold and D. Friedman. Battling infectious diseases in the 20th century: the impact of vaccines. *The Wall Street Journal.* http://graphics.wsj.com/infectious-diseases-and-vaccines.

[175] T. Kekeritz. Weather Radials: An infographic on heat waves and snow storms in 35 cities around the globe. http://weather-radials.com.

[176] E.-J. Marey, *La méthode graphique dans les sciences expérimentales et principalement en physiologie et en médecine.* G. Masson, 1878.

[177] R. Metoyer, B. Lee, N. Henry Riche, and M. Czerwinski, *Understanding the verbal language and structure of end-user descriptions of data visualizations*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1659–1662, 2012.

[178] J. Brosz, M. A. Nacenta, R. Pusch, S. Carpendale, and C. Hurter, *Transmogrification: Causal manipulation of visualizations*, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 97–106, 2013.

[179] "Microsoft Power BI: Custom visuals." `https://powerbi.microsoft.com/en-us/custom-visuals`.

[180] "TypeScript: JavaScript that scales." `https://www.typescriptlang.org/`, accessed Dec. 3th, 2016.

[181] "React: A JavaScript library for building user interfaces." `https://facebook.github.io/react/`, accessed Dec. 3th, 2016.

[182] "WebAssembly." `https://webassembly.org`.

[183] "Flux." `https://facebook.github.io/flux/docs/overview.html`.

[184] G. J. Badros, A. Borning, and P. J. Stuckey, *The Cassowary linear arithmetic constraint solving algorithm*, *ACM Transactions on Computer-Human Interaction (TOCHI)* **8** (2001), no. 4 267–306.

[185] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[186] S. K. Card, T. P. Moran, and A. Newell, *The keystroke-level model for user performance time with interactive systems*, *Communications of the ACM* **23** (1980), no. 7 396–410.

[187] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko, *Data Illustrator: Gallery*, 2018. `http://data-illustrator.com/gallery.php`.

[188] O. Axelsson, *Iterative solution methods*. Cambridge University Press, 1996.

[189] H. Xia, N. Riche, F. Chevalier, B. D. Araujo, and D. Wigdor, *DataInk: Enabling direct and creative data-oriented drawing*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 223:1–223:13, 2018.

[190] D. Holten, *Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data*, *IEEE Transactions on Visualization and Computer Graphics* **12** (2006), no. 5 741–748.

[191] T. Dwyer, K. Marriott, and M. Wybrow, *Integrating edge routing into force-directed layout*, in *International Symposium on Graph Drawing*, pp. 8–19, 2006.

[192] M. Conlen and J. Heer, *Idyll: A markup language for authoring and publishing interactive articles on the web*, in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, (New York, NY, USA), pp. 977–989, ACM, 2018.

[193] M. Zöckler, D. Stalling, and H.-C. Hege, *Interactive visualization of 3D-vector fields using illuminated steam lines*, *Proceedings of IEEE Visualization '96* (1996) 107–114.

[194] A. Simon, R. C. Smith, and R. R. Pawlicki, *Omnistereo for panoramic virtual environment display systems*, in *Proceedings of IEEE Virtual Reality (VR)*, pp. 67–73, IEEE, 2004.

[195] T. A. DeFanti, A. Saad, G. Wickham, and R. A. Ainsworth, *Luxor – Temple – Night*, in *CAVEcam Virtual Reality Photography Collection.* UC San Diego Library Digital Collections, 2016.

[196] "PhaseSpace Impulse X2 motion tracking system." `http://www.phasespace.com/impulse-motion-capture.html`, accessed September 2015.

[197] C. Lee, S. Bonebrake, D. Bowman, and T. Höllerer, *The role of latency in the validity of ar simulation*, in *Proceedings of IEEE Virtual Reality (VR)*, pp. 11–18, 2010.

[198] T. Chandler, M. Cordeil, T. Czauderna, T. Dwyer, J. Glowacki, C. Goncu, M. Klapperstueck, K. Klein, K. Marriott, F. Schreiber, *et. al.*, *Immersive analytics*, in *Big Data Visual Analytics (BDVA), 2015*, pp. 1–8, IEEE, 2015.

[199] M. Cordeil, T. Dwyer, K. Klein, B. Laha, K. Marriot, and B. H. Thomas, *Immersive collaborative analysis of network connectivity: CAVE-style or head-mounted display?*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2016), no. 1 441–450.

[200] O.-H. Kwon, C. Muelder, K. Lee, and K.-L. Ma, *A study of layout, rendering, and interaction methods for immersive graph visualization*, *IEEE Transactions on Visualization and Computer Graphics* **22** (2016), no. 7 1802–1815.

[201] "C3.js: D3-based reusable chart library." `http://c3js.org/`, accessed Dec. 3, 2016.

[202] "NVD3: Reusable charts for D3.js." `http://nvd3.org/`, accessed Dec. 3, 2016.

[203] M. Romero, "SVG performance test." `http://bl.ocks.org/mjromper/95fef29a83c43cb116c3`, accessed Dec. 3, 2016.

[204] B. McDonnel and N. Elmqvist, *Towards utilizing GPUs in information visualization: A model and implementation of image-space operations*, *IEEE Transactions on Visualization and Computer Graphics* **15** (2009), no. 6 1105–1112.

[205] S. Haroz, R. Kosara, and S. L. Franconeri, *ISOTYPE Visualization: Working memory, performance, and engagement with pictographs*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '15, pp. 1191–1200, ACM, 2015.

[206] L. Wilkinson, *Dot plots*, *The American Statistician* **53** (1999), no. 3 276–281.

[207] D. A. Keim, M. C. Hao, U. Dayal, and M. Hsu, *Pixel bar charts: A visualization technique for very large multi-attribute data sets*, *Information Visualization* **1** (2002), no. 1 20–34.

[208] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams, *Squares: Supporting interactive performance analysis for multiclass classifiers*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 1 61–70.

[209] "The daily routines of famous creative people." `https://podio.com/site/creative-routines/`, accessed Dec. 3, 2016.

[210] Y. Hu, *Efficient, high-quality force-directed graph drawing*, *Mathematica Journal* **10** (2005), no. 1 37–71.

[211] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, *Nature* **324** (1986), no. 6096 446–449.

[212] S. McKenna, N. H. Riche, B. Lee, J. Boy, and M. Meyer, *Visual narrative flow: Exploring factors shaping data visualization story reading experiences*, *Computer Graphics Forum (EuroVis '17)* **36** (2017), no. 3 377–387.

[213] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert, *Altair: Interactive statistical visualizations for python*, *The Journal of Open Source Software* **3** (2018), no. 32.

[214] R. Cabello *et. al.*, "three.js: JavaScript 3D library." `https://threejs.org/`.

[215] Gapminder Foundation, *Gapminder*, 2009. `http://www.gapminder.org`.

[216] *HoloJS: A framework for creating holographic apps using JavaScript and WebGL*, 2018. `https://github.com/Microsoft/HoloJS`, accessed Apr. 5th, 2019.

[217] G. Grinstein, K. Cook, P. Havig, K. Liggett, B. Nebesh, M. Whiting, K. Whitley, and S. Knoecni, *VAST 2011 challenge: Cyber security and epidemic*, *IEEE VAST* **2011** (2011) 299–301. http://www.cs.umd.edu/hcil/varepository/benchmarks.php#VAST2011, accessed August 2018.

[218] Microsoft, *Timeline Storyteller*, 2017. https://timelinestoryteller.com.

[219] H. Mei, Y. Ma, Y. Wei, and W. Chen, *The design space of construction tools for information visualization: A survey*, *Journal of Visual Languages & Computing* (2017).

[220] A. Kirk, *The Chartmaker Directory*, 2017. http://chartmaker.visualisingdata.com.

[221] C. Plaisant, *The challenge of information visualization evaluation*, in *Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI)*, pp. 109–116, 2004.

[222] S. Carpendale, *Evaluating information visualizations*, in *Information visualization*, pp. 19–45. Springer, 2008.

[223] G. Robertson, M. Czerwinski, D. Fisher, and B. Lee, *Selected human factors issues in information visualization*, *Reviews of Human Factors and Ergonomics* **5** (2009), no. 1 41–81.

[224] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale, *Empirical studies in information visualization: Seven scenarios*, *IEEE Transactions on Visualization and Computer Graphics* **18** (2012), no. 9 1520–1536.

[225] F. Amini, M. Brehmer, G. Bolduan, C. Elmer, and B. Wiederkehr, *Evaluating data-driven stories & storytelling tools*, in *Data-Driven Storytelling* (N. Henry Riche, C. Hurter, N. Diakopoulos, and S. Carpendale, eds.). A K Peters/CRC Press, 2018.

[226] A. Satyanarayan and J. Heer, *Authoring narrative visualizations with Ellipsis*, *Computer Graphics Forum* **33** (2014), no. 3.

[227] D. Schroeder and D. F. Keefe, *Visualization-by-sketching: An artist's interface for creating multivariate time-varying data visualizations*, *IEEE Transactions on Visualization and Computer Graphics* **22** (2016), no. 1 877–885.

[228] G. G. Méndez, U. Hinrichs, and M. A. Nacenta, *Bottom-up vs. top-down: Trade-offs in efficiency, understanding, freedom and creativity with InfoVis tools*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 841–852, 2017.

[229] M. Brehmer, S. Carpendale, B. Lee, and M. Tory, *Pre-design empiricism for information visualization: scenarios, methods, and challenges*, in *Proceedings of the ACM Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*, pp. 147–151, 2014.

[230] F. Amini, N. Henry Riche, B. Lee, C. Hurter, and P. Irani, *Understanding data videos: Looking at narrative visualization through the cinematography lens*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1459–1468, 2015.

[231] "Definition of *benchmark* by Merriam-Webster." `https://www.merriam-webster.com/dictionary/benchmark`.

[232] J.-D. Fekete and C. Plaisant, *InfoVis 2003 Contest: Visualization and pair wise comparison of trees*, 2003. `http://www.cs.umd.edu/hcil/iv03contest`.

[233] C. Plaisant, *Information visualization benchmarks repository*, 2003. `http://www.cs.umd.edu/hcil/InfovisRepository`.

[234] C. Plaisant, *Visual analytics benchmark repository*, 2006. `https://www.cs.umd.edu/hcil/varepository`.

[235] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko, *vispubdata.org: A metadata collection about IEEE visualization (VIS) publications*, *IEEE Transactions on Visualization and Computer Graphics* **23** (2017), no. 9 2199–2206.

[236] P. Isenberg, T. Isenberg, M. Sedlmair, J. Chen, and T. Möller, *Toward a deeper understanding of visualization through keyword analysis*, Tech. Rep. RR-8580, INRIA, France, 2014.

[237] H.-J. Schulz, *Treevis.net: A tree visualization reference*, *IEEE Computer Graphics & Applications (CG&A)* (2011), no. 6 11–15.

[238] S. Ribecca, "The data visualisation catalogue." `https://datavizcatalogue.com/`.

[239] S. McKenna, N. Henry Riche, B. Lee, J. Boy, and M. Meyer, *Visual narrative flow: Exploring factors shaping data visualization story reading experiences*, *Computer Graphics Forum* **36** (2017), no. 3 377–387.

[240] C. D. Stolper, B. Lee, N. Henry Riche, and J. Stasko, *Data-driven storytelling techniques: Analysis of a curated collection of visual stories*, in *Data-Driven Storytelling* (N. Henry Riche, C. Hurter, N. Diakopoulos, and S. Carpendale, eds.). A K Peters/CRC Press, 2018.

[241] C. M. Danis, F. B. Viegas, M. Wattenberg, and J. Kriss, *Your place or mine?: Visualization as a community component*, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 275–284, 2008.

[242] F. B. Viegas, M. Wattenberg, M. McKeon, F. Van Ham, and J. Kriss, *Harry Potter and the meat-filled freezer: A case study of spontaneous usage of visualization tools*, in *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, pp. 159–159, 2008.

[243] B. Shneiderman and C. Plaisant, *Strategies for evaluating information visualization tools: Multi-dimensional in-depth long-term case studies*, in *Proceedings of the ACM Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*, pp. 1–7, 2006.

[244] M. Sedlmair, M. Meyer, and T. Munzner, *Design study methodology: Reflections from the trenches and the stacks*, *IEEE Transactions on Visualization and Computer Graphics* **18** (2012), no. 12 2431–2440.

[245] M. Brehmer, S. Ingram, J. Stray, and T. Munzner, *Overview: The design, adoption, and analysis of a visual document mining tool for investigative journalists*, *IEEE Transactions on Visualization and Computer Graphics* **20** (2014), no. 12 2271–2280.