Authoring and Publishing Interactive Articles

Matthew Conlen

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

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Jeffrey Heer, Chair

Jon Froehlich

Tim Althoff

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Authoring and Publishing
Interactive Articles

Matthew Conlen

Chair of the Supervisory Committee:
Jerre D. Noe Endowed Professor Jeffrey Heer
Paul G. Allen School of Computer Science & Engineering

Interactive articles—a dynamic medium that combines narrative text with interactive graphics and computational simulations—are important for education, journalism, and scientific publishing because they can improve learning outcomes by promoting active engagement in readers. However, interactive articles are difficult and costly to produce: they are created using complex general-purpose programming tools and authors receive little guidance on effective design, due in part to a lack of empirical evidence of usage patterns in real-world situations. This research supports the design, creation, publication, and evaluation of interactive articles by presenting analysis of interactive article design, three authoring tools that lower the threshold to create interactive articles, and systems that facilitate the collection and analysis of article usage data.

This dissertation starts with an analysis of sixty interactive articles, in which we connect interactive article design patterns to research in multimedia learning, digital journalism, and human-computer interaction. We identify a set of five key affordances of the medium for educational and journalistic usage and discuss implications for authoring tools. Building on this foundation, we present *Idyll*, an expressive domain-specific markup language that reduces the amount of code and effort needed to create and publish interactive articles through a reactive document model and standard component library. Since our initial deployment, *Idyll* has been used by educators, journalists, and researchers to, for example,

create interactive materials for an undergraduate physics curriculum, communicate the results of detailed urban development models, and publish interactive data-driven analyses in college newspapers. However, the system is too technical and low-level for some users.

We extend *Idyll* with two higher-level authoring systems: *Idyll Studio* is a WYSIWYG-style structured editor that lowers the threshold for authors by replacing general-purpose programming tools with an easy-to-learn interface centered on direct and instrumental manipulation; *Fidyll* is a simplified markup language that can be used to rapidly generate interactive articles as well as static PDFs, slideshows, and videos from a single source file, improving the portability, archivability, and accessibility of interactive articles while limiting the amount of markup needed overall. Leveraging *Idyll*'s reactive document model, we develop a system that automates the detailed instrumentation of interactive articles and publish an open dataset of more than 50,000 session logs from in-the-wild usage of three interactive articles. To facilitate learning from such data, we develop two corresponding visual analysis tools that aid in inspecting usage data at both micro- and macro-levels.

Taken together, this suite of tools serves as a set of modular building blocks for authoring and publishing interactive articles that communicate information effectively. The software in this dissertation has been used by educators, journalists, and researchers to create articles that have resonated with hundreds of thousands of readers, that were covered in digital design publications, that are among the top online search results for their respective mathematical and statistical topics, that have gone viral online, and that received honor at the IEEE VIS Workshop on Visualization for AI Explainability. The code is open-source and can be found at `https://github.com/idyll-lang`. Examples can be found at `https://idyll-lang.org/gallery`.

TABLE OF CONTENTS

Page

List of Figures iii

Chapter 1: Introduction 1
1.1 Research Contributions 2
1.2 Prior Publications and Authorship 5

Chapter 2: Background 6
2.1 Narrative Visualization 6
2.2 Multimedia Learning 8
2.3 Domain-Specific Languages 9
2.4 Computational Documents 10

Chapter 3: Communicating with Interactive Articles 13
3.1 Interactive Articles: Theory & Practice 14
3.2 Challenges for Authoring Interactives 33
3.3 Critical Reflections 36

Chapter 4: *Idyll*: A Markup Language for Interactive Articles 40
4.1 Related Work 42
4.2 Design Requirements 47
4.3 The *Idyll* Language 50
4.4 Examples 56
4.5 Deployment 62
4.6 Discussion 67

Chapter 5: *Idyll Studio*: Structured Editing of Interactive Articles 69
5.1 Motivation & Related Work 71
5.2 *Idyll Studio* 76
5.3 Evaluation: First-Use Study 82
5.4 Evaluation: Expressiveness 90

5.5 Future Work 92

Chapter 6: *Fidyll*: Cross-Format Data Stories & Explorable Explanations 94
6.1 Background 96
6.2 Formative Interviews 98
6.3 Motivating Scenarios 102
6.4 *Fidyll* 103
6.5 Evaluation 112
6.6 Discussion 116

Chapter 7: Capture & Analysis of Active Reading Behaviors 118
7.1 Motivation & Related Work 120
7.2 Case Study Article Design 123
7.3 Tools for Capturing Reader Activity 126
7.4 Tools for Analyzing Reader Activity 127
7.5 Case Study Article Analysis 131
7.6 Discussion 142

Chapter 8: Conclusion 146
8.1 Summary of Contributions 146
8.2 Future Research Directions 148
8.3 Concluding Remarks 152

Bibliography 153

LIST OF FIGURES

Figure Number Page

3.1 Exemplary interactive articles from around the web. In the interactive version of this figure, readers can select an article thumbnail to view additional details about the article. 13

3.2 Interactive articles are applicable to variety of domains, such as research dissemination, journalism, education, and policy and decision making. In the interactive version of this figure, readers could tab through sections to view details about specific application areas on demand. 15

3.3 We identified five affordances of interactive articles as a medium for communicating complex information. 16

3.4 In the interactive version of this figure, readers can click the play button or scrub over the video frames to watch and control the animation. 17

3.5 In "Extensive Data Shows Punishing Reach of Racism for Black Boys" [36], the use of unit animation carries the main visualization of the story to highlight real people's lives changing over time. 18

3.6 In "Cutthroat Capitalism: The Game" [303], readers play the role of a pirate commander, giving them a unique look at the economics that led to rise in piracy off the coast of Somalia. 19

3.7 In the interactive version of this figure, readers can drag the sliders to change the number of boids in the simulation and adjust the different parameters to find interesting configurations. 21

3.8 In "Teachable Machines" [302], a reader uses their own live video camera to train a machine learning image classifier in-browser without any extra computational resources. 22

3.9 In "Visualizing Quaternions" [252], a viewer can take control of an interactive video while narration continues in the background. 24

3.10 In the interactive version of this figure, readers can click and drag to make a guess of the data's trend over time. Afterward, the real data will be revealed. 25

3.11 In "The Gyllenhaal Experiment" [121], readers are tasked to type the names of celebrities with challenging spellings. After submitting a guess, a visualization shows the reader's entry against everyone else's, scaled by the frequency of different spellings. 26

3.12 In "How To Remember Anything Forever-ish" [72], readers use spaced repetition to learn about spaced repetition. 27

3.13 In "How Much Hotter Is Your Hometown Than When You Were Born?" [234], a reader enters their birthplace and birth year and is shown multiple visualizations describing the impact of climate on their hometown. 28

3.14 In "Quantum Country" [193], the interactive textbook uses spaced repetition and allows a reader to opt-in and save their progress while reading through dense material and mathematical notion over time. 29

3.15 In the interactive version of this figure, readers can click any point to listen to a different bird's chirp. 31

3.16 In the interactive version of this figure, readers can choose between 1 of 4 machine-generated images and brush over the circle callouts to display a short message about each region. Generated images from [151, 152] 32

3.17 In the interactive version of this figure, readers can click to reveal what each mark of notation or variable represents in the equation. 33

3.18 In the interactive version of this figure, readers can drag the slider to display the theorem's statement in increasing levels of detail. 34

3.19 The "Myth of the Impartial Machine" [109] was one of five articles published in The Parametric Press. The article used techniques like animation, data visualizations, explanatory diagrams, margin notes, and interactive simulations to explain how biases occur in machine learning systems. 37

3.20 Interactive communication opportunities from both research and practice. . . 38

4.1 *Idyll* been used to create a variety articles—which have been visited over half a million times—including *Beginner's Guide to Dimensionality Reduction* [85], Best Paper Honorable Mention at the Workshop on Visualization for AI Explainability at IEEE VIS. 40

4.2 An *Idyll* article that interactively links text with visual demonstrations to explain the Barnes-Hut approximation for simulating physical forces. The visualization on the right keeps a fixed position, updating as the user progresses through the article and interacts with the text. 41

4.3 Comparing the readability and expressiveness of web-based publishing tools. Many languages focus on providing a legible authoring experience, or on facilitating the implementation of complex designs. *Idyll* aims to achieve high readability of its text, while maintaining expressiveness. It does this by imposing a clean, structured interface between text and interactive elements, providing a standard library, and by allowing users to write custom code where necessary. 43

4.4 A stereotypical behavior of interactive documents: values can be modified by the reader in order to effect change change elsewhere on the page. In this example, adapted from the Tangle's documentation [293], the number of snacks consumed can be modified, and the sentence updates to show the equivalent amount of calories. 49

4.5 The *Idyll* article model. (1) The *Idyll* compiler transforms input markup into a list of document nodes; (2) each of these nodes has a dictionary of properties that determine its behavior (optionally including a list of children which are recursively rendered). (3) The nodes are combined with theme and layout information to (4) construct a static HTML page. When the page is loaded in a browser it is (5) hydrated with event handlers and a reactive state to drive interface updates. 52

4.6 In *Idyll*, a document's variable values can be bound to control widgets, allowing authors to quickly implement components that respond to user input. This graphic shows how reader interaction propagates through *Idyll*'s state and affects the rendered output. 53

4.7 An interactive article explaining Kernel Density Estimation. The article uses scroll-based interactivity defined declaratively in *Idyll* markup. Listing 4.4 shows the *Idyll* markup used to specify this layout. The equations and controls shown in this screenshot were defined in *Idyll* markup and reactively parameterize the custom graphic. 59

4.8 *The Etymology of Trig Functions*, an interactive blog post written with Idyll. This example integrates narrative and graphics through user interaction with the text. As a user hovers their mouse over stylized text, the graphic on the right updates to show a geometric interpretation of the concept being discussed. 60

4.9 Usage counts of various language features across student groups, along with the total unique uses of features per group. The students were comfortable using the Markdown syntax, and relied heavily on *Idyll*'s standard library of components, reducing the overall amount of code that they had to write. . . . 63

4.10 A student-authored *Idyll* article explaining the conflict-driven clause learning (CDCL) SAT solver. The article displays custom visualizations parameterized by *Idyll* variables and standard components. The detail shown above visualizes a logical formula; readers can use the buttons to toggle the truth assignment of variables, and see the effects on clauses and the overall formula. 64

4.11 This student article, *A* Search and Dijkstra's Algorithm*, is motivated by the use of path-finding algorithms in video games. The students developed a custom game, controlled via *Idyll* components, in which players choose optimal search parameters in order to win. 65

4.12 A student article explaining the Travelling Salesman Problem. The students leveraged *Idyll*'s features for modifying styles and layout. They were able to create a highly personalized page, incorporating both scroll- and step-based navigation. 67

5.1 *Idyll Studio* builds on the *Idyll* markup language by creating a structured editing interface for creating, manipulating, and publishing interactive articles. We implement a tree expansion algorithm to transform arbitrary *Idyll* programs into structured editing interfaces. *Text editors* allow an author to edit and style text while *component editors* allow for orchestration of component behaviors. *Insertion points* can be used to add new components or text blocks to the article via drag-and-drop. 69

5.2 *Idyll Studio* utilizes direct and instrumental manipulation to let authors compose and choreograph interactive and data-driven articles. The interface consists of a live article view (A) and an editing panel (B) on the left-hand side. Authors can use text editors (C), insertion points (D), and component editors (E) to construct their articles. The variables panel (F) lets the author inspect and modify the document's reactive state, and document (G) and component (H) styles can be applied to customize the aesthetics. 75

5.3 Two examples of how *Idyll Studio* interoperates with existing tools: (A) users can click "edit" to call up their system text editor and modify a component's source code; any changes made are immediately reflected in the article view; (B) similarly, users can edit media assets like images or SVGs by invoking a domain-specific editor like Figma and see changes immediately reflected in context. 81

5.4 The distribution of self-reported technical expertise of our user study participants by category: developer tools, web technologies, JavaScript libraries, markup languages, and visualization grammars. Overlaid vertical lines convey the mean value. 83

5.5 Our first-use study asked participants to complete a series of tasks that involved adding interactivity and data-driven elements to their articles. The tasks were chosen to represent common design patterns found in interactive articles. 84

5.6 Specification of a *Scroller* layout, which displays a sequence of text blocks over a visualization that updates as a reader scrolls. Text is edited in place and the chart's specification is parameterized with a reactive variable that tracks scroll depth. 91

5.7 A view of *The Barnes-Hut Approximation*, an existing explorable explanation reimplemented using *Idyll Studio*. A graphic stays fixed to the right side of the page as readers scroll, and responds to links and input widgets placed in the text. 91

6.1 *Fidyll* supports serializing five different output formats from a single source document. Here different versions of the case study *Quantifying Political Ideology* are shown. The different formats each have their own strengths and weaknesses. 94

6.2 *Fidyll*'s data model centers on the notion of giving readers a tour through a high-dimensional parameter space. Each *scene* defines its own parameter space, and each *stage* within the scene defines a set of parameter values. *Controls* give readers leeway to explore parts of this space that aren't directly visited by a stage. 104

6.3 *Fidyll*'s software architecture. Authors write narrative text in a markup format and provide parameterized graphics. The markup is parsed and normalized into our schema, which then is used to collect parameter values, generate code, audio, and graphics, and ultimately produce various formats which can be published on the web or elsewhere. 107

6.4 We used *Fidyll* to create three interactive articles. In the interactive article version of *Quantifying Political Ideology*, sections of text scroll on the left half of the screen (A) while graphics remain fixed in place on the right (B); graphics update as the reader scrolls and respond to interactions with controls embedded in the text. The presentation version of *Climate and Economic Modeling* provides high-level text summaries of visualizations and embedded controls (C), which can be used to manipulate full-screen graphics (D). In the low-motion HTML version of *Fast & Accurate Gaussian Kernel Density Estimation*, graphics are repeatedly embedded in a single text column with various parameters (E); readers can still manipulate controls to further explore the parameter space (F). 111

7.1 An online interactive article *How To: Tune A Guitar* (left) and visualizations of collected reader activity data. *HopScroll* (center) visualizes reader progress over time, revealing reading patterns and fixation points. *Readuction* (right) uses dimensionality reduction of reader feature vectors to enable nuanced segment analysis; linked views show timing and event information for selected points. Along with these tools we present Idyll language extensions for automating the collection of detailed log data, and discuss reading patterns discovered. 118

7.2 *Beat Basics* was produced by Megan Vo, an undergraduate computer science student. She designed it to teach curious readers the basics of rhythmic time signatures in music. 120

7.3 The *Beginner's Guide to Dimensionality Reduction* uses artworks from the Metropolitan Museum of Art as examples for introducing dimensionality reduction techniques. 124
7.4 Scroll positions over time for *How To: Tune A Guitar*. Plots highlight exemplary patterns: (A) *Preview & Read*, (B) *Super Tuners*, (C) *Scroll & Bounce*, and (D) *Balanced Engagement*. 128
7.5 An annotated UMAP projection of *Beat Basics* reader sessions. We used the Readuction tool to identify cohorts of readers exhibiting similar behavior by observing the distributions of feature vectors for subsets of reader sessions. The feature vectors consist of counts of variable changes (a proxy for engagement with interactive widgets) and time spent on each section of content. 132
7.6 Example scroll paths from *Beat Basics*. (A) shows a reader who backtracks in the content several times; (B) a reader progresses linearly before scrolling down quickly to preview a section and subsequently spend time reading it; (C) shows a reader who read through the content forward and then in reverse. 133
7.7 The distribution of minimum values of a range slider in *The Beginner's Guide to Dimensionality Reduction*. The slider started in the far right position. Many users did not engage with it; others moved it all the way to the left. . . 137
7.8 Example scroll paths from the *Beginner's Guide to Dimensionality Reduction*. (A) shows a reader who skimmed the beginning of the article, but spent more time in the later sections; (B) shows a reader who quickly reached the end of the article, spent time there, and then went back to re-read earlier content. . 138
7.9 UMAP projection of feature vectors for *Beginner's Guide to Dimensionality Reduction* readers. The projection reveals broad clusters of readers, such as those who consume all the content, those who leave the page quickly, those who engage heavily with article features, and those read the content multiple times. 139
7.10 UMAP projection of feature vectors for readers of *How To: Tune a Guitar*. This article offered the most polished mobile experience of the three, and we see a less stark difference in the distributions between the desktop and mobile readers. Through interactive use of our Readuction tool we identify clusters of users who exhibit interesting engagement patterns. 141

ACKNOWLEDGMENTS

I would like to thank the mentors and colleagues I learned from and worked with at the University of Washington, including Jeffrey Heer, Jessica Hullman, Arvind Satyanarayan, Leilani Battle, Jon Froehlich, Dominik Moritz, Kanit Wongsuphasawat, Alex Kale, Michael Correll, Eunice Jun, Tongshuang Wu, Yang Liu, Megan Vo, Alan Tan, and Manesh Jhawar.

The editors, authors, and supporters of the *Parametric Press* published incredible work beyond any of my expectations. Thank you to Fred Hohman, Sara Stalla, Andrew Odewahn for nurturing this project, to Andrew Sass for beautiful design, and to Omar Shehata, Alice Feng, Shuyan Wu, Alyson Powell Key, James McGirk, Riccardo Maria Bianchi, Aatish Bhatia, Geoffrey Litt, Seth Thompson, Benjamin Cooley, Christina Orieschnig, Halden Lin, Aishwarya Nirmal, Shobhit Hathi, and Lilian Liang for telling inspiring interactive stories.

Thank you to the Explorable Explanations community for being a constant source of inspiration and feedback, including Nicky Case, Srini Kadamati, Hamish Todd, Toph Tucker, Chris Makler, Amanda Curtis, Adam Pearce, and many others. Thank you to the talented journalists, designers, and programmers that I learned from and worked with along the way, including Sam Petulla, Reuben Fischer-Baum, Gus Wezerek, Kevin Quealy, Wilson Andrews, Archie Tse, Kate LaRue, and Walt Hickey. Thank you to Scott Davidoff, Hillary Mushkin, Maggie Hendrie, and Santiago Lombeyda for mentorship during my time at NASA JPL, and for teaching me the power of design thinking and trusting design processes.

Thank you to Albert Wenger for being one of the first people to recognize the vision of the Idyll project and for being one of its earliest supporters, and to my former colleagues at Rhizome, including Zachary Kaplan and Dragan Espenschied, for their kindness, thoughtfulness, and continued support. Thank you to Andrew Osheroff, Jeremy Freeman, and Deep Ganguli for their collaboration, friendship, and encouragement during this time. Thank you to Julia Powers for steadfastly enduring my time as a graduate student.

DEDICATION

In memory of Keith.

Chapter 1

INTRODUCTION

Computing has changed how people communicate. The transmission of news, messages, and ideas is instant. Anyone's voice can be heard. Access to digital communication technologies such as the Internet is so fundamental to daily life that their disruption by government is condemned by the United Nations Human Rights Council [217]. But while the technology to distribute our ideas has grown in leaps and bounds, the interfaces have remained largely the same. Parallel to the development of the internet, researchers like Alan Kay and Douglas Engelbart worked to build technology that would empower individuals and enhance cognition. Kay imagined the Dynabook [157] in the hands of children across the world. Engelbart, while best remembered for his "mother of all demos," was more interested in the ability of computation to augment human intellect [108]. Later designs pointed to a future where computers are connected and assist people in decision-making and communicating using rich graphics and interactive user interfaces [105]. While some technologies have seen mainstream adoption, such as Hypertext [212], unfortunately, many others have not. The most popular publishing platforms, like WordPress and Medium, prioritize social features and ease-of-use while limiting the ability for authors to use the dynamic features of the web.

In the spirit of previous computer-assisted cognition technologies, a new type of computational communication medium has emerged that leverages active reading techniques to make ideas more accessible to a broad range of people. These interactive articles build on a long history, from Plato [284] to PHeT [95] to explorable explanations [291]. They have been shown to be more engaging [124], can help improve recall and learning [194], and attract broad readership and acclaim—for example, some of the *New York Times* [156, 60] and the *Washington Post*'s [272] most read stories are interactive articles—yet our knowledge of effective interactive article design is limited, and we do not have many effective tools to support their creation.

While interactive articles can be an effective medium of communication in domains like education [194], journalism [261, 124], and scientific publishing [102], they are created using complex general-purpose programming tools making them difficult and costly to produce. Furthermore, authors receive little guidance on effective article design, exacerbated by a lack of empirical evidence of usage patterns in real-world situations. My research aims to support the design, authoring, and publication of interactive articles by building a better understanding of common design techniques, creating high-level authoring tools, and building systems to support article evaluation.

1.1 Research Contributions

The chapters of this dissertation can be organized in three parts: *designing*, *authoring*, and *evaluating*. In Chapter 2, I lay out the existing research in this space, drawing on research from the human-computer interaction, information visualization, digital journalism, and multimedia learning communities, among others, and follow that with a survey of interactive article usage in Chapter 3, which identifies common design techniques and affordances of the medium, including a discussion of the implications for authoring tools and highlights where more research needs to be done. I developed three authoring tools, the first of which is *Idyll* (Chapter 4), a highly expressive tool that reduces the amount of code needed to create and publish interactive articles through a simple markup language, reactive document model, and component library. The open source tool has been used to create a variety of articles, which have been read by hundreds of thousands of people; however, the system is too technical and low-level for some users. To address this, I present two extensions to the system: a structured editor, *Idyll Studio* (Chapter 5), lowers the barrier to entry to create and publish interactive articles and allows experts to iterate on designs more rapidly; *Fidyll* (Chapter 6) supports authoring interactive articles through a higher-level language, reducing the amount of non-narrative markup that authors must write to produce accessible and archivable interactive articles across multiple formats. In Chapter 7, I present an automated process to collect high-level *in-the-wild* article usage data, and show how this data can be used to create user models which are assessed in two novel visual analytics tools.

The Design of Interactive Articles. This dissertation starts with an overview of prior academic work, and then contributes a survey of over sixty interactive articles, in which I tie together the theory and practice of creating interactive articles. and identify key affordances of the medium. I identify how the format can be used for *Connecting People and Data*, *Making Systems Playful*, *Prompting Self-Reflection*, *Personalizing Reading*, and *Reducing Cognitive Load*, allowing authors to communicate complex data-driven technical information in a format that improves engagement, learning, and recall. In this section I identify challenges and opportunities associated with authoring and publishing interactive articles that serve as motivation for the subsequent chapters.

Tools for Authoring Interactive Articles. While there are many reasons that interactive articles are beneficial for readers, they are also difficult for authors to produce. Creating an Interactive article requires a broad range of skills, often including a large amount of general-purpose programming, and frequently involves collaboration between contributors with vastly different areas of expertise. I develop authoring tools that limit the amount of markup needed to specify article behavior, broadly reduce the need for general purpose programming tools, facilitate collaboration, and promote publishing articles in multiple formats to support archivability and accessibility of the content.

Idyll. This style of interactive media has the potential to engage a large audience and more clearly explain concepts, but is expensive and time consuming to produce. Drawing on industry experience and interviews with domain experts, I contribute design tools to make it easier to author and publish interactive articles. I introduce *Idyll*, a novel "compile-to-the-web" language for web-based interactive narratives. *Idyll* implements a flexible article model, allowing authors control over document style and layout, reader-driven events (such as button clicks and scroll triggers), and a structured interface to JavaScript components. Through examples and first-use results from undergraduate computer science students, I show how *Idyll* reduces the amount of effort and code required to create interactive articles.

Idyll Studio. Interactive articles are an effective medium of communication in education, journalism, and scientific publishing, yet are created using complex general-purpose programming tools. I present *Idyll Studio*, a structured editor for authoring and publishing interactive and data-driven articles. I extend the *Idyll* framework to support reflective

documents, which can inspect and modify their underlying program at runtime, and show how this functionality can be used to reify the constituent parts of a reactive document model—components, text, state, and styles—in an expressive, interoperable, and easy-to-learn graphical interface. In a study with 18 diverse participants, all could perform basic editing and composition, use datasets and variables, and specify relationships between components. Most could choreograph interactive visualizations and dynamic text, although some struggled with advanced uses requiring unstructured code editing. My findings suggest *Idyll Studio* lowers the threshold for non-experts to create interactive articles and allows experts to rapidly specify a wide range of article designs.

Fidyll. Narrative visualization is a powerful communicative technique that can take on various formats such as interactive articles, slideshows, and data videos. These formats each have their strengths and weaknesses, but existing authoring tools only support one output target. I conducted a series of formative interviews with seven domain experts to understand needs and practices around multi-platform narrative visualizations, and developed *Fidyll*, a cross-genre compiler for authoring interactive data stories and explorable explanations. Our open-source tool can be used to rapidly create visual narratives across a variety of genres including static articles, interactive articles, slideshows, and videos. I evaluate the system through a series of real-world-based usage scenarios, showing how it benefits authors in the domains of data journalism, scientific publishing, and nonprofit advocacy. *Fidyll* provides expressive leverage to authors, reducing the amount of non-narrative markup needed for cross-format data stories by 80-90%, allowing them to focus on their domain specific content.

Evaluating Interactive Article Usage. Popular analytics tools provide only coarse information about how readers interact with individual pages, and laboratory studies often fail to capture the variability of a real-world audience. I contribute extensions to the *Idyll* markup language to automate the detailed instrumentation of interactive articles and corresponding visual analysis tools for inspecting reader behavior at both micro- and macro-levels. I present three case studies of interactive articles that were instrumented, posted online, and promoted via social media to reach broad audiences, and share data from over 50,000 reader sessions. The tools are used to characterize article-specific interaction patterns, compare behavior across desktop and mobile devices, and reveal reading patterns

common across articles. The findings, tools, and corpus of behavioral data can help advance and inform more comprehensive studies of narrative visualization.

1.2 Prior Publications and Authorship

While I am the primary author of the research that is included in this dissertation, it would be inaccurate to present this as solitary work. The following chapters are the result of years of collaboration between myself and my advisor Jeffrey Heer, members of the University of Washington Interactive Data Lab, open source contributors to the *Idyll* project, and others. All of the work was done in collaboration with Professor Heer over the course of my Ph.D. at the University of Washington. *Idyll* (Chapter 4), was first introduced at the 2017 Computation+Journalism Symposium [83] and then evolved before publication at ACM UIST 2018 [84], and was influenced by data-oriented publishing tools that I developed while working as a data journalist at *FiveThirtyEight*. I extended this work to support collection and analysis of interactive article usage data (Chapter 7), published at EuroVis 2019 [88]; this was done in collaboration with Alex Kale, who contributed to the analysis tools, and featured, as case studies, interactive articles created in collaboration Fred Hohman and Alex Kale and an interactive article by Megan Vo. One of these articles, *Beginner's Guide to Dimensionality Reduction*, also appeared at the first Workshop on Visualization for AI Explainability at IEEE VIS 2018 [85] where it received an honorable mention for Best Paper. The survey of interactive articles and their affordances (Chapter 3) was initially published in 2020 in the online journal Distill [141] in collaboration with Fred Hohman and Polo Chau. This work was the result of a long collaboration between Fred and myself, stemming from a rejected CHI workshop paper, and includes a section reflecting on our work on the *Paramtric Press*, an interactive digital magazine edited by Fred Hohman, Sara Stalla, and myself; the *Paramtric Press* was also presented at the VisComm Workshop at IEEE VIS 2019 [86]. *Idyll Studio* (Chapter 5) was developed in collaboration with Megan Vo and Alan Tan who assisted in implementing the system; the paper appeared at ACM UIST 2021 [90]. I will use the first person plural throughout these chapters to acknowledge these contributions. At the time of this writing, *Fidyll* (Chapter 6), is under review at EuroVis 2022.

Chapter 2
BACKGROUND

Interactive articles sit at the intersection of several areas of research including information visualization, human-computer interaction, education, digital publishing, and programming languages. In this chapter, we discuss related work on narrative information visualization, multimedia learning, domain specific programming languages, and authoring and publishing systems for computational documents, including code notebooks and other similar environments. In Chapter 3, we look at an in-depth survey of applications and design techniques relating to interactive articles specifically.

2.1 Narrative Visualization

In foundational work on narrative visualization, Segel & Heer [262] investigated the techniques that are used in data-driven storytelling, looking at stories that have been published by news outlets, researchers, and in business reports, and articulated a design space of narrative visualization, including a set of seven genres of narrative visualization (magazine style, annotated chart, partitioned poster, flow chart, comic strip, slide show, and film/video/animation). A challenge for researchers is that the tools and techniques used by practitioners are constantly shifting. Both Stolper et al. [273] and McKenna et al. [198] refine Segel & Heer's design space, updating it to reflect changes in practice and incorporating new techniques like interactive articles which use scroll-based triggers [198].

Researchers have noted several opportunities for work to be done in this space [177, 181], including evaluating the effectiveness of data-driven storytelling techniques, and building tools that help authors use these techniques in their stories. The work presented in this dissertation tackles some of this, including presenting new authoring tools (Chapters 4–6), tools to support the collection and analysis of reading behaviors to support evaluation of article designs (Chapter 7).

2.1.1 Applications & Design Techniques

The information visualization research community has examined the role of narrative in explanatory visualizations, including storytelling approaches [262, 273, 118], the effects of sequence [143] and framing [142], as well as in-depth analysis of usage of different narrative visualization formats such as data comics [32, 34], video [24, 58], and interactive articles [198, 314]. The use of interactivity and visualization in writing extends beyond journalism, for example into scientific publishing [189, 292]. Victor has written persuasively in favor of adding certain types of interactivity to writing that would traditionally be presented as static text [290]. His essay *Explorable Explanations* [291] illustrates some of the types of interactions that we seek to enable with our authoring systems. Notably, Distill [99] is an online machine learning journal with interactive articles and seeks to bridge the gap between techniques common to news media and academic publishing. Olah, an editor of Distill, wrote on the value of using interactivity to explain complex topics, arguing that interactive publishing platforms like Distill could expedite the dissemination of new research ideas [219]. Distill publishes posts that utilize visualizations, animations, and linked parameters in text to explain machine learning research. Distill also publishes the source code for their articles, which reveals just how much work goes into creating them: individual articles often require several hundred custom code commits, made over a period of months [220].

Interactive articles are useful because they support animation and interactivity, allowing authors to take advantage of multimedia learning techniques [194] (explored in-depth in Chapter 3). However, they may not be preferable in all situations or contexts. Researchers have found that individual preferences play a key role in audience engagement, for example McKenna et al. [198] found that some readers prefer slideshows to scroll-based articles and engaged more if articles were presented in the format of their preference. Interactivity and animation also present challenges in accessibility: some readers of websites which utilize scroll-based interactivity and parallax will experience motion sickness [113]; interactive graphics require additional development to support screen readers and may not be beneficial for visually impaired users [163]. Additionally interactive articles present a challenge for preservation and archiving; while preservation tools [160] and formats [204] for

rich interactive web content exist, they are seldom incorporated as part of the publishing process [62] and are not as well supported as static document formats like PDF [122].

2.1.2 Authoring Tools

Lee et al. [181] articulated a model of the visual data storytelling process. They describe how multiple collaborators (e.g., data analyst, scripter, and editor) work together to create and present data-driven stories in professional settings, which serves as a guideline for the different roles and tasks that authoring tools should support. For example, editors need to be able to easily edit the text of stories and arrange graphical components without needing to understand how the components are implemented. Scripters need to be able to rapidly iterate on component implementations and see how changes look in context.

A number of systems have been developed to facilitate authoring narrative visualizations or interactive articles [200, 254, 293, 179, 218, 275]. Ellipsis, for example, provides a DSL and graphical interface for building narrative visualizations; TimelineStoryteller [61] is a tool for building narrative timelines. However, existing systems are either limited in their expressivity or require that authors use general purpose programming tools. For example, some existing WYSYIWG-style editors allow users to insert arbitrary interactive components into web-based articles [66] but, in these systems, components are treated as independent black boxes and cannot communicate amongst one another—a requirement for common design patterns like linked text and graphics [314]. On the other hand, code-based approaches, while sufficiently expressive, have a high threshold for productive use and are often inappropriate for non-technical users like some journalists and educators [179].

2.2 Multimedia Learning

Researchers in the field of multimedia learning have extensively examined the combined use of audio, video, text, and graphics and have identified a set of educational principles of multimedia learning as well as limitations [194], and communicative data visualization has also been framed as a learning problem [20]. Interactive articles are used by authors to produce educational materials that are more effective for learners compared to static alternatives. In Chapter 3, we present an in-depth survey of interactive article designs and

connect them to these principles. In the future, these principles could be embedded into authoring tools and be used to help provide automated design assistance.

Evaluating the educational effectiveness of dynamic material can be difficult. For example, research into the effectiveness of animation to facilitate learning showed that many studies showing a positive learning effects may just be due to differences in content between animated and static versions of materials [283]; however, later research showed that animated transitions can significantly improve graphical perception for depicting transitions between data graphics [134]. Simple aspects of article design such as use of ambient videos or even choice of color can impact reader affect; a positive affect in readers can lead to more time spent engaging with the article [124], and improved learning outcomes [285].

2.3 Domain-Specific Languages

Domain-specific languages (DSLs) are important for program authors because they can improve expressiveness or ease-of-use compared to general-purpose programming languages [201]. Several DSLs developed for use specifying interactive information visualizations have found wide adoption ([55, 259]). We develop DSLs for authoring interactive articles: in Chapter 4, we introduce *Idyll*, an expressive DSL for interactive articles; in Chapter 6, we introduce *Fidyll*, a higher-level DSL that compiles to *Idyll* and is used to facilitate the production of cross-format interactive visual narratives. These languages are informed by prior work on functional reactive programming (FRP) and literate programming.

2.3.1 Functional Reactive Programming

FRP [37] is a programming paradigm where authors specify the program's behavior in relation to a stream of input events, making it a natural fit for programming user interfaces. Flapjax [202] is a language that brings FRP to the web. Flapjax showed that event-driven reactivity is a natural fit for web applications. FRP has also been shown to be an effective means for enabling expressive declarative languages in the domains of animation [107] and visualization [259] specification, and can be similarly applied in this domain. React [4] is a JavaScript library for declaratively building user interfaces, centered around reactive components. Our domain-specific language *Idyll* is implemented using React and uses an

event-driven reactive parameterization of article specifications to allow authors to concisely express a wide range of program behavior for interactive articles.

2.3.2 Literate Programming

Literate programming [174] languages allow program authors to mix executable text with prose. Environments like Codestrates [239] and Leisure [65] offer a similar editing experience to computational notebooks but offer additional flexibility—for example the ability to customize the user interface—through the ability to use the programming environment to modify its own user interface. By making the entire medium dynamic, literate programs can "blur the line" between authoring environment and application [173]; this technique has been extended to data visualization for ubiquitous analytics [35]. Our domain-specific languages *Idyll* and *Fidyll* are related to literate programming in that they are markup languages that mix text and embedded logic. However, in contrast to literate programs in which the prose is typically documentation supporting the code, our languages are used to specify rich visual stories to be read by an audience who isn't interested in reading the underlying code implementation, only the graphics that the code produces.

2.4 Computational Documents

Computation media, broadly construed, includes diverse formats like video games [51], electronic literature [17], and interactive documentaries [28]. Here we provide background on computational documents as they relate to data scientists and communicators.

2.4.1 Computational Notebooks

Computational notebook environments such as Jupyter [226], Observable [215], and Mathematica [307], are frequently used by programmers and data scientists to create and share graphics and computations. Like Observable, *Idyll* (Chapter 4) leverages reactive semantics. These environments typically target exploratory analysis use cases in which an analyst uses the notebook as a feature-rich REPL to interactively construct visualizations and data transformations to support their analytic needs [227] in a shareable and collaborative for-

mat [300]. While interactive articles are related to computational notebooks, our work focuses on the creation of active reading experiences as opposed to interfaces for data analytics. Whereas notebooks focus on the coding experience using an interface consisting of a linear list of code and output cells, our systems allow authors to customize nearly every aspect of the look-and-feel of the output including layout, navigation, and styling.

2.4.2 Hypertext

The web is the dominant publishing platform today, and offers authors the ability to create highly interactive documents [48]. There have been many efforts to make it easier to write for the web using a simpler and more concise syntax than HTML, for example, Markdown [125] is a popular markup language designed to be fast to write and easy to read. Jekyll [2] is a blog engine that allows users to write posts in plain text or Markdown files and deploy them to the web. Visdown [149] extends the idea of compiling Markdown to HTML pages by allowing authors to specify data visualizations in their markup. A user can provide a declarative Vega-Lite [257] specification directly in the Markdown file, and this is used to render a chart in the final markup. Creating *interactive* documents involves writing custom JavaScript and HTML. It can become difficult to balance the narrative portion of a project with the low-level details of code. To this end, the *New York Times* developed ArchieML [274], a markup language for editors to work with text that will subsequently be embedded in an interactive article. These markup languages, however, provide little or no support for logic that tightly couples the text and graphics, limiting the expressivity, and limiting authors' ability to create interactive links between text and visualizations [314].

Several research systems facilitate end-user creation of data-driven websites. These tools are aimed at a more general use-case than our work, but are nonetheless informative when considering programs whose primary *output* are web pages. Gneiss [75] is an application that provides a drag-and-drop interface with which users can create GUI components. These components may draw their value from cells on a spreadsheet which may act as a proxy to an arbitrary REST service. Gneiss lets users pick from a predefined set of UI elements, but does not provide a compelling way for users to embed their own custom elements. Mavo [289] is an

abstraction on top of HTML and introduces syntax that allows users to bind data directly to templated HTML. The tool allows users to author full data-driven websites without relying on a database service. Other web frameworks utilize a similar component-oriented architecture, for example Polymer [230] is a JavaScript library for building applications using Web Components, a new standard being added to web browsers. These frameworks simplify web development, however they still require users to navigate complex application code. *Idyll* intentionally separates JavaScript code and editorial copy, so that non-technical users may still make simple edits to the text of an article without navigating complex code.

2.4.3 Interactive Graphics

Graphics are often created using DSLs like D3 [55] or Processing [242], visualization grammars such as Vega [258] or Vega-Lite [257], or libraries like Stardust [244] and regl [188] that offer APIs to utilize GPUs for rendering large amounts of data in two or three dimensions. Because of the breadth of tools used to create interactive graphics on the web and the diversity of formats used to represent them, our work on authoring systems takes a "bring your own graphics" approach, offering systems that are used to orchestrate the behavior of web-based parameterized graphics [317] regardless of how they were created.

Several graphical tools have been developed to support the design of data visualization via a direct manipulation graphical interface [256], including Data Illustrator [185] and Charticulator [243]. Lyra [255, 316] allows users to specify interactive visualizations by example, while Kitty [158] and Apparatus [260] support the authoring of interactive diagrams through sketching and direct manipulation, respectively. Several other tools offer graphical interfaces for the design of specialized graphics. For example, Apparatus [260] is an editor for creating interactive diagrams; TextAlive [153] is an integrated design environment for creating kinetic typography videos. There tools are complementary to the work in this dissertation, as we focus on the larger document structure and providing a generic API for interactive interactive graphics and allow users to plug-in graphics created with any of these systems. Our authoring tools—*Idyll*, *Idyll Studio*, and *Fidyll*—are designed to supplement, rather than supplant, these tools.

Chapter 3

COMMUNICATING WITH INTERACTIVE ARTICLES

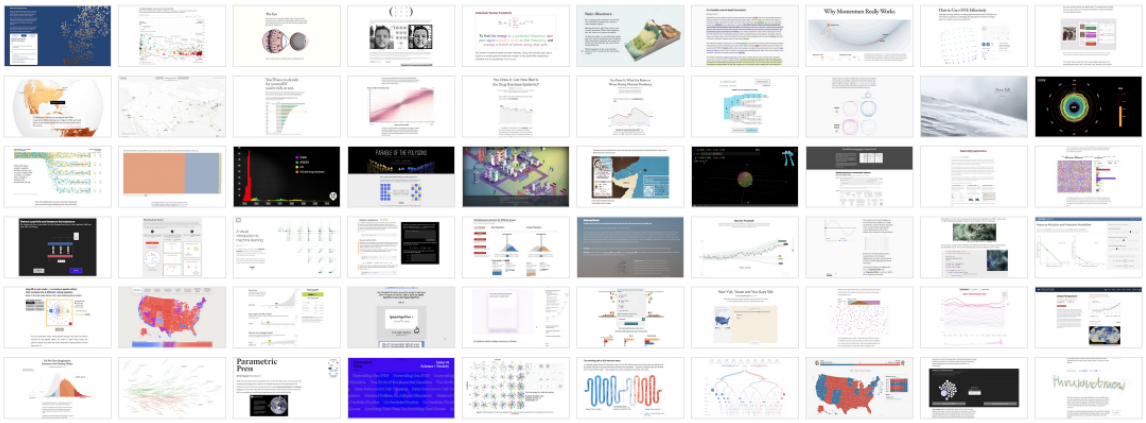


Figure 3.1: Exemplary interactive articles from around the web. In the interactive version of this figure, readers can select an article thumbnail to view additional details about the article.

Today there is a growing excitement around the use of interactive articles for communication since they offer unique capabilities to help people learn and engage with complex ideas that traditional media lacks. In this chapter we connect the dots between interactive articles such as those featured in this journal and publications like the *New York Times* and the techniques, theories, and empirical evaluations put forth by academic researchers across the fields of education, human-computer interaction, information visualization, and digital journalism. We show how digital designers are operationalizing these ideas to create interactive articles that help boost learning and engagement for their readers compared to static alternatives. After describing the affordances of interactive articles, we provide critical reflections from our own experience with open-source, interactive publishing at scale. We conclude with discussing practical challenges and open research directions for authoring,

designing, and publishing interactive articles.

This style of communication—and the platforms which support it—are still in their infancy. When choosing where to publish this work, we wanted the medium to reflect the message. This article was originally published in the online machine learning journal *Distill* and included a number of interactive figures. Those figures are reproduced here in a static form, however, they were developed specifically to highlight the advantages of interactive digital media, and we highly recommend viewing them in their original format online at `https://distill.pub/2020/communicating-with-interactive-articles/`. Journals like *Distill* are not only pushing the boundaries of machine learning research but also offer a space to put forth new interfaces for dissemination. This work ties together the theory and practice of authoring and publishing interactive articles. It demonstrates the power that the medium has for providing new representations and interactions to make systems and ideas more accessible to broad audiences.

3.1 Interactive Articles: Theory & Practice

Interactive articles draw from and connect many types of media, from static text and images to movies and animations. But in contrast to these existing forms, they also leverage interaction techniques such as details-on demand, belief elicitation, play, and models and simulations to enhance communication.

While the space of possible designs is far too broad to be solved with one-size-fits-all guidelines, by connecting the techniques used in these articles back to underlying theories presented across disparate fields of research we provide a missing foundation for designers to use when considering the broad space of interactions that could be added to a born-digital article.

We draw from a corpus of sixty interactive articles to highlight the breadth of techniques available and analyze how their authors took advantage of a digital medium to improve the reading experience along one or more dimensions, for example, by reducing the overall cognitive load, instilling positive affect, or improving information recall.

Because diverse communities create interactive content, this medium goes by many different names and has not yet settled on a standardized format nor definition. Researchers

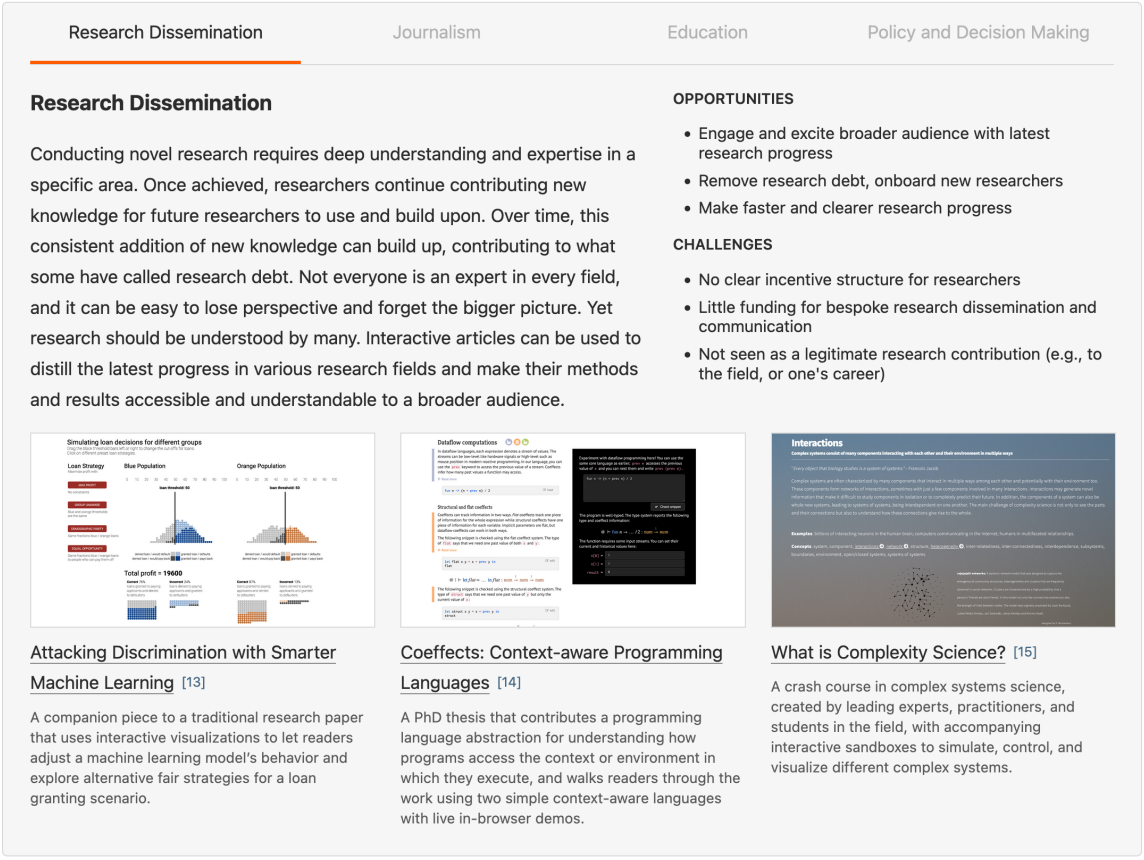


Figure 3.2: Interactive articles are applicable to variety of domains, such as research dissemination, journalism, education, and policy and decision making. In the interactive version of this figure, readers could tab through sections to view details about specific application areas on demand.

have proposed artifacts such as explorable multiverse analyses [102], explainables [1], and exploranations [311] to more effectively disseminate their work, communicate their results to the public, and remove research debt [219]. In newsrooms, data journalists, developers, and designers work together to make complex news and investigative reporting clear and engaging using interactive stories [181]. Educators use interactive textbooks as an alternative learning format to give students hands-on experience with learning material [253].

Besides these groups, others such as academics, game developers, web developers, and

Figure 3.3: We identified five affordances of interactive articles as a medium for communicating complex information.

designers blend editorial, design, and programming skills to create and publish explorable explanations [291], interactive fiction [17], interactive non-fiction [266], active essays [308], and interactive games [52]. While these all slightly differ in their technical approach and target audience, they all largely leverage the interactivity of the modern web.

We focus on five unique affordances of interactive articles, listed below. In-line videos and example interactive graphics are presented alongside this discussion to demonstrate specific techniques.

Connecting People and Data

As visual designers are well aware, and as journalism researchers have confirmed empirically [124], an audience which finds content to be aesthetically pleasing is more likely to have a positive attitude towards it. This in turn means people will spend more time engaging with content and ultimately lead to improved learning outcomes. While engagement itself may not be an end goal of most research communications, the ability to influence both audience attitude and the amount of time that is spent is a useful lever to improve learning: we know from education research that both time spent [114] and emotion [285] are predictive of learning outcomes.

Animations can also be used to improve engagement [25]. While there is debate amongst researchers if animations in general are able to more effectively convey the same information compared to a well designed static graphic [283], animation has been shown to be effective

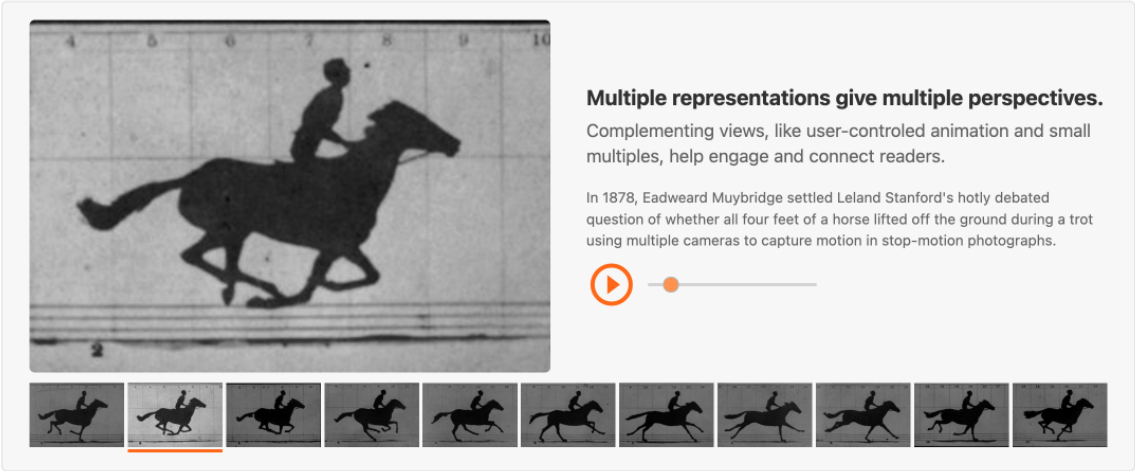


Figure 3.4: In the interactive version of this figure, readers can click the play button or scrub over the video frames to watch and control the animation.

specifically for communicating state transitions [134], uncertainty [144], causality [203], and constructing narratives [278]. A classic example of this is Muybridge's motion study [207] that can be seen in Figure 3.4: while the series of still images may be more effective for answering specific questions like, "Does a horse lift all four of its feet off the ground when it runs?" watching the animation in slow motion gives the viewer a much more visceral sense of how it runs. A more modern example can be found in OpenAI's reporting on their hide-and-seek agents [38]. The animations here instantly give the viewer a sense of how the agents are operating in their environment.

Passively, animation can be used to add drama to a graphic displaying important information, but which readers may otherwise find dry. Scientific data which is inherently time varying may be shown using an animation to connect viewers more closely with the original data, as compared to seeing an abstracted static view. For example, Ed Hawkins designed "Climate Spirals," which shows the average global temperature change over time [129]. This presentation of the data resonated with a large public audience, so much so that it was displayed at the opening ceremony at the 2016 Rio Olympics. In fact, many other climate change visualizations of this same dataset use animation to build suspense

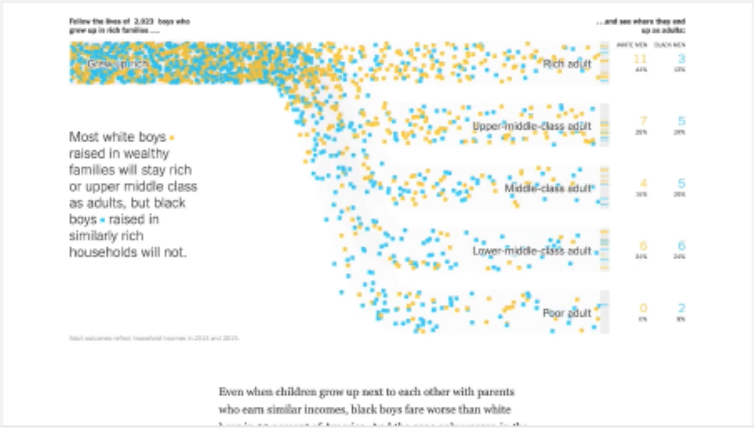


Figure 3.5: In "Extensive Data Shows Punishing Reach of Racism for Black Boys" [36], the use of unit animation carries the main visualization of the story to highlight real people's lives changing over time.

and highlight the recent spike in global temperatures [249, 241, 210, 233].

By adding variation over time, authors have access to a new dimension to encode information and an even wider design space to work in. Consider the animated graphic in the *New York Times* story "Extensive Data Shows Punishing Reach of Racism for Black Boys," which shows economic outcomes for 10,000 men who grew up in rich families [36]. While there are many ways in which the same data could have been communicated more succinctly using a static visualization [93], by utilizing animation, it became possible for the authors to design a unit visualization in which each data point shown represented an individual, reminding readers that the data in this story was about real peoples' lives.

Unit visualizations have also been used to evoke empathy in readers in other works covering grim topics such as gun deaths [73] and soldier deaths in war [127]. Using person-shaped glyphs (as opposed to abstract symbols like circles or squares) has been shown not to produce additional empathic responses [57], but including actual photographs of people helps readers gain interest in, remember [53, 54], and communicate complex phenomena [267] using visualizations. Correll argues that much of the power of visualization comes from abstraction, but quantization stymies empathy [92]. He instead suggests anthropo-

Figure 3.6: In "Cutthroat Capitalism: The Game" [303], readers play the role of a pirate commander, giving them a unique look at the economics that led to rise in piracy off the coast of Somalia.

morphizing data, borrowing journalistic and rhetoric techniques to create novel designs or interventions to foster empathy in readers when viewing visualizations [92, 146].

Regarding the format of interactive articles, an ongoing debate within the data journalism community has been whether articles which utilize scroll-based graphics (scrollytelling) are more effective than those which use step-based graphics (slideshows). McKenna et al. [198] found that their study participants largely preferred content to be displayed with a step- or scroll-based navigation as opposed to traditional static articles, but did not find a significant difference in engagement between the two layouts. In related work, Zhi et al. found that performance on comprehension tasks was better in slideshow layouts than in vertical scroll-based layouts [314]. Both studies focused on people using desktop (rather than mobile) devices. More work is needed to evaluate the effectiveness of various layouts on mobile devices, however the interviews conducted by MckEnna et al. suggest that additional features, such as supporting navigation through swipe gestures, may be necessary to facilitate the mobile reading experience.

The use of games to convey information has been explored in the domains of journalism [52] and education [271]. Designers of newsgames use them to help readers build empathy

with their subject, for example in *The Financial Times's* "Uber Game" [50], and explain complex systems consisting of multiple parts, for example in *Wired's* "Cutthroat Capitalism: The Game" [303]. In educational settings the use of games has been shown to motivate students while maintaining or improving learning outcomes [296].

As text moves away from author-guided narratives towards more reader-driven ones [262], the reading experience becomes closer to that of playing a game. For example, the critically acclaimed explorable explanation "Parable of the Polygons" puts play at the center of the story, letting a reader manually run an algorithm that is later simulated in the article to demonstrate how a population of people with slight personal biases against diversity leads to social segregation [128].

Making Systems Playful

Interactive articles utilize an underlying computational infrastructure, allowing authors editorial control over the computational processes happening on a page. This access to computation allows interactive articles to engage readers in an experience they could not have with traditional media. For example, in "Drawing Dynamic Visualizations", Victor demonstrates how an interactive visualization can allow readers to build an intuition about the behavior of a system, leading to a fundamentally different understanding of an underlying system compared to looking at a set of static equations [295]. These articles leverage active learning and reading, combined with critical thinking [21] to help diverse sets of people learn and explore using sandboxed models and simulations [291].

Complex systems often require extensive setup to allow for proper study: conducting scientific experiments, training machine learning models, modeling social phenomenon, digesting advanced mathematics, and researching recent political events, all require the configuration of sophisticated software packages before a user can interact with a system at all, even just to tweak a single parameter. This barrier to entry can deter people from engaging with complex topics, or explicitly prevent people who do not have the necessary resources, for example, computer hardware for intense machine learning tasks. Interactive articles drastically lower these barriers.

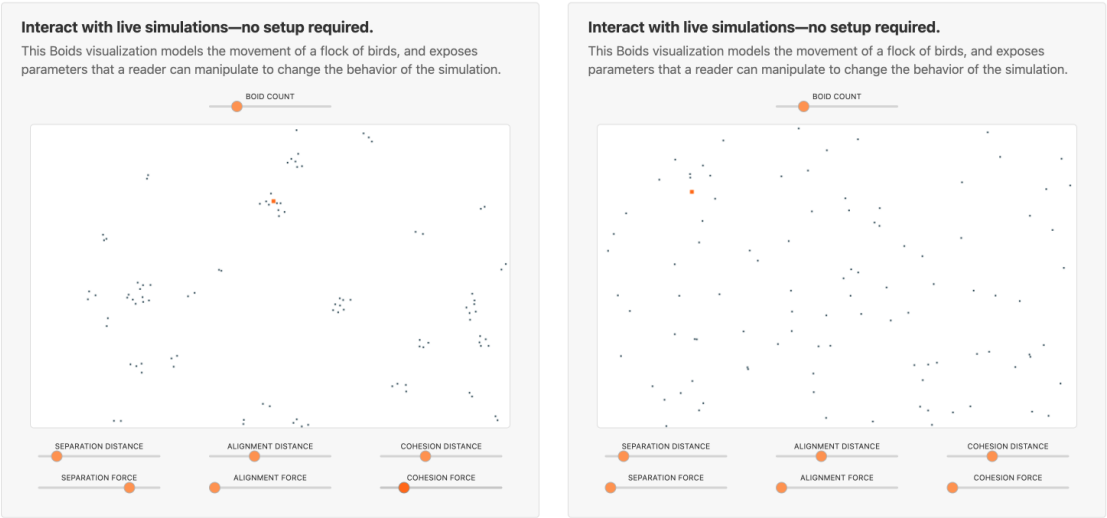


Figure 3.7: In the interactive version of this figure, readers can drag the sliders to change the number of boids in the simulation and adjust the different parameters to find interesting configurations.

Science that utilizes physical and computational experiments requires systematically controlling and changing parameters to observe their effect on the modeled system. In research, dissemination is typically done through static documents, where various figures show and compare the effect of varying particular parameters. However, efforts have been made to leverage interactivity in academic publishing, summarized in [102]. Reimagining the research paper with interactive graphics [292], as exploranations [311], or as explorable multiverse analyses [102], gives readers control over the reporting of the research findings and shows great promise in helping readers both digest new ideas and learn about existing fields that are built upon piles of research debt [219].

Beyond reporting statistics, interactive articles are extremely powerful when the studied systems can be modeled or simulated in real-time with interactive parameters without setup, e.g., in-browser sandboxes. Consider the example in Figure 3.7 of a Boids simulation that models how birds flock together. Complex systems such as these have many different parameters that change the resulting simulation. These sandbox simulations allow readers

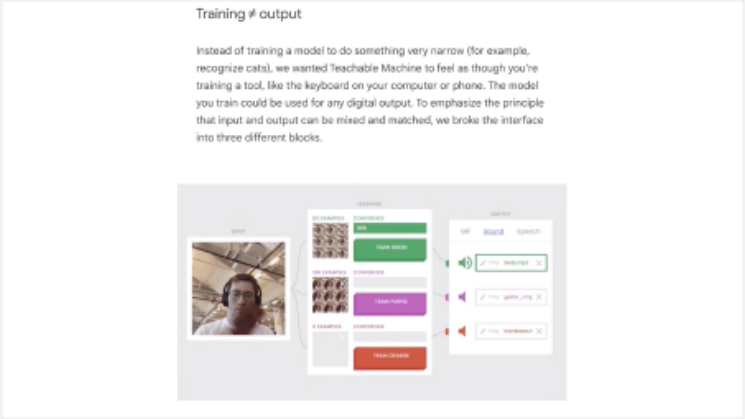


Figure 3.8: In "Teachable Machines" [302], a reader uses their own live video camera to train a machine learning image classifier in-browser without any extra computational resources.

to play with parameters to see their effect without worrying about technical overhead or other experimental consequences.

This is a standout design pattern within interactive articles, and many examples exist ranging in complexity. "How You Will Die" visually simulates the average life expectancy of different groups of people, where a reader can choose the gender, race, and age of a person [309]. "On Particle Physics" allows readers to experiment with accelerating different particles through electric and magnetic fields to build intuition behind electromagnetism foundations such as the Lorentz force and Maxwell's equations—the experiments backing these simulations cannot be done without multi-million dollar machinery [49]. "Should Prison Sentences Be Based On Crimes That Haven't Been Committed Yet?" shows the outcome of calculating risk assessments for recidivism where readers adjust the thresholds for determining who gets parole [40].

The dissemination of modern machine learning techniques has been bolstered by interactive models and simulations. Three articles, "How to Use t-SNE Effectively" [301], "The Beginner's Guide to Dimensionality Reduction" [85], and "Understanding UMAP" [81] show the effect that hyperparameters and different dimensionality reduction techniques have on creating low dimensional embeddings of high-dimensional data. A popular approach is to

demonstrate how machine learning models work with in-browser models [269], for example, letting readers use their own video camera as input to an image classification model [302] or handwriting as input to a stroke prediction model [67]. Other examples are aimed at technical readers who wish to learn about specific concepts within deep learning. Here, interfaces allow readers to choose model hyperparameters, datasets, and training procedures that, once selected, visualize the training process and model internals to inspect the effect of varying the model configuration [268, 148].

Interactive articles commonly communicate a single idea or concept using multiple representations. The same information represented in different forms can have different impact. For example, in mathematics often a single object has both an algebraic and a geometric representation. A clear example of this is the definition of a circle [68]. Both are useful, inform one another, and lead to different ways of thinking. Examples of interactive articles that demonstrate this include various media publications' political election coverage that break down the same outcome in multiple ways, for example, by voter demographics, geographical location, and historical perspective [265, 154, 5].

The Multimedia Principle states that people learn better from words and pictures rather than words or pictures alone [194], as people can process information through both a visual channel and auditory channel simultaneously. Popular video creators such as 3Blue1Brown [251] and Primer [135] exemplify these principles by using rich animation and simultaneous narration to break down complex topics. These videos additionally take advantage of the Redundancy Principle by including complementary information in the narration and in the graphics rather than repeating the same information in both channels [196].

While these videos are praised for their approachability and rich exposition, they are not interactive. One radical extension from traditional video content is also incorporating user input into the video while narration plays. A series of these interactive videos on "Visualizing Quaternions" lets a reader listen to narration of a live animation on screen, but at any time the viewer can take control of the video and manipulate the animation and graphics while simultaneously listening to the narration [252].

Utilizing multiple representations allows a reader to see different abstractions of a single idea. Once these are familiar and known, an author can build interfaces from multiple

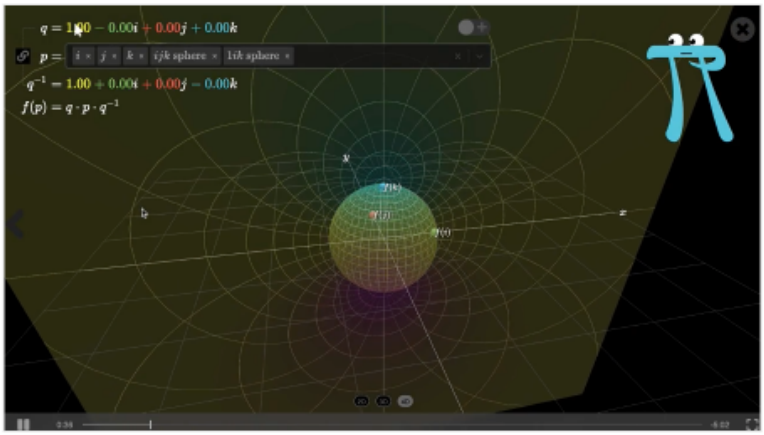


Figure 3.9: In "Visualizing Quaternions" [252], a viewer can take control of an interactive video while narration continues in the background.

representations and let readers interact with them simultaneously, ultimately leading to interactive experiences that demonstrate the power of computational communication mediums. Next, we discuss such experiences where interactive articles have transformed communication and learning by making live models and simulations of complex systems and phenomena accessible.

Prompting Self-Reflection

Asking a student to reflect on material that they are studying and explain it back to themselves—a learning technique called self-explanation—is known to have a positive impact on learning outcomes [78]. By generating explanations and refining them as new information is obtained, it is hypothesized that a student will be more engaged with the processes which they are studying [77]. When writing for an interactive environment, components can be included which prompt readers to make a prediction or reflection about the material and cause them to engage in self-explanation [167].

While these prompts may take the form of text entry or other standard input widgets, one of the most prominent examples of this technique used in practice comes from the *New York Times* "You Draw It" visualizations [23, 155, 64]. In these visualizations, readers are

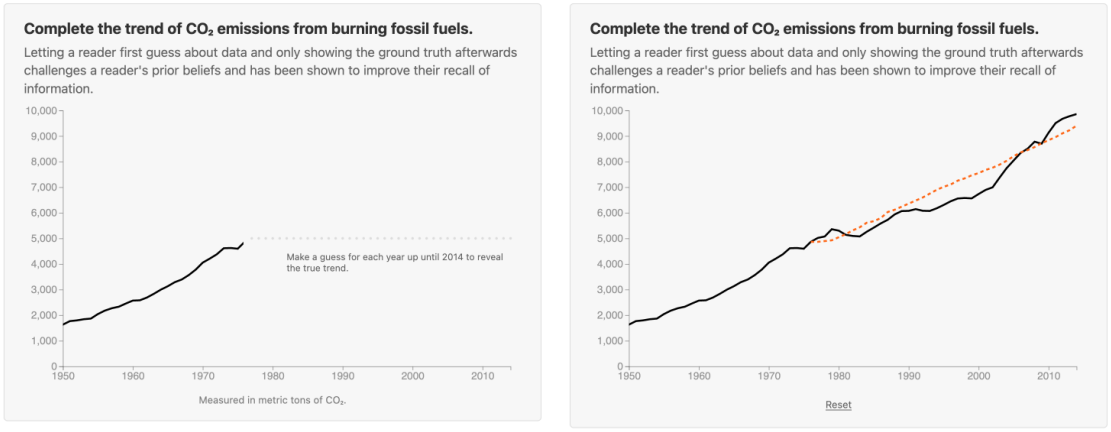


Figure 3.10: In the interactive version of this figure, readers can click and drag to make a guess of the data's trend over time. Afterward, the real data will be revealed.

prompted to complete a trendline on a chart, causing them to generate an explanation based on their current beliefs for why they think the trend may move in a certain direction. Only after readers make their prediction are they shown the actual data. Kim et al. showed that using visualizations as a prompt is an effective way to encourage readers to engage in self explanation and improve their recall of the information [167, 213]. Figure 3.10 shows one these visualizations for carbon dioxide emissions from burning fossil fuels. After clicking and dragging to guess the trend, your guess will be compared against the actual data.

In the case of "You Draw It," readers were also shown the predictions that others made, adding a social comparison element to the experience. This additional social information was not shown to necessarily be effective for improving recall [166]. However, one might hypothesize that this social aspect may have other benefits such as improving reader engagement, due to the popularity of recent visual stories using this technique, for example in *The Pudding's* "Gyllenhaal Experiment" [121] and *Quartz's* "How do you draw a circle?" [126].

Prompting readers to remember previously presented material, for example through the use of quizzes, can be an effective way to improve their ability to recall it in the future [117].

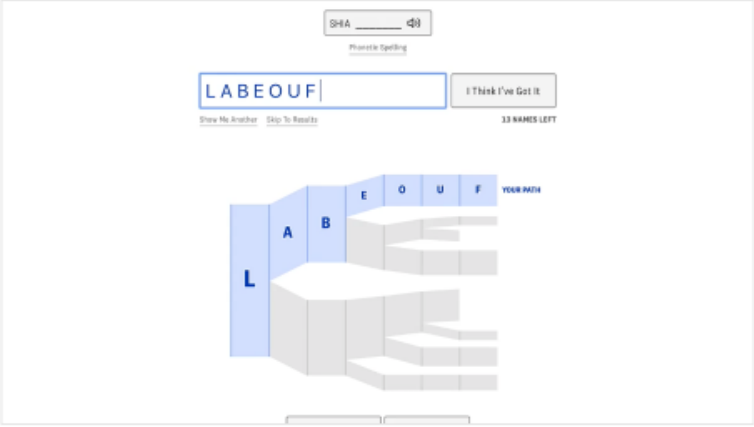


Figure 3.11: In "The Gyllenhaal Experiment" [121], readers are tasked to type the names of celebrities with challenging spellings. After submitting a guess, a visualization shows the reader's entry against everyone else's, scaled by the frequency of different spellings.

This result from cognitive psychology, known as the testing effect [247], can be utilized by authors writing for an interactive medium [3]. While testing may call to mind stressful educational experiences for many, quizzes included in web articles can be low stakes: there is no need to record the results or grade readers. The effect is enhanced if feedback is given to the quiz-takers, for example by providing the correct answer after the user has recorded their response [39].

The benefits of the testing effect can be further enhanced if the testing is repeated over a period of time [150], assuming readers are willing to participate in the process. The idea of spaced repetition has been a popular foundation for memory building applications, for example in the Anki flash card system. More recently, authors have experimented with building spaced repetition directly into their web-based writing [72, 193], giving motivated readers the opportunity to easily opt-in to a repeated testing program over the relevant material.

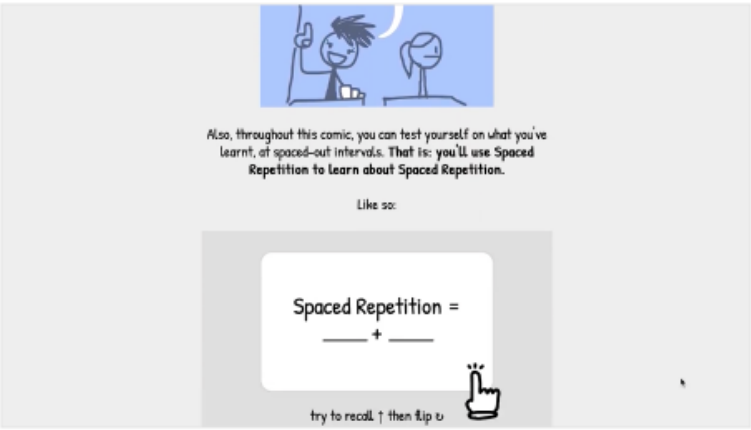


Figure 3.12: In "How To Remember Anything Forever-ish" [72], readers use spaced repetition to learn about spaced repetition.

Personalizing Reading

Content personalization—automatically modifying text and multimedia based on a reader's individual features or input (e.g., demographics or location)—is a technique that has been shown to increase engagement and learning within readers [91] and support behavioral change [98]. The PersaLog system gives developers tools to build personalized content and presents guidelines for personalization based on user research from practicing journalists [19]. Other work has shown that "personalized spatial analogies," presenting distance measurements in regions where readers are geographically familiar with, help people more concretely understand new distance measurements within news stories [164].

Personalization alone has also been used as the standout feature of multiple interactive articles. Both "How Much Hotter Is Your Hometown Than When You Were Born?" [234] and "Human Terrain" [96] use a reader's location to drive stories relating to climate change and population densities respectively. Other examples ask for explicit reader input, such as a story that visualizes a reader's net worth to challenge a reader's assumptions if they are wealthy or not (relative to the greater population) [236], or predicting a reader's political party affiliation [79]. Another example is the interactive scatterplot featured in "Find Out

Figure 3.13: In "How Much Hotter Is Your Hometown Than When You Were Born?" [234], a reader enters their birthplace and birth year and is shown multiple visualizations describing the impact of climate on their hometown.

If Your Job Will Be Automated" [305]. In this visualization, professions are plotted to determine their likelihood of being automated against their average annual wage. The article encourages readers to use the search bar to type in their own profession to highlight it against the others.

An interactive medium has the potential to offer readers an experience other than static, linear text. Non-linear stories, where a reader can choose their own path through the content, have the potential to provide a more personalized experience and focus on areas of user interest [266]. For example, the *BBC* has used this technique in both online articles [187] and in a recent episode of "Click" [45], a technology focused news television program. Non-linear stories present challenges for authors, as they must consider the myriad possible paths through the content, and consider the different possible experiences that the audience would have when pursuing different branches.

Another technique interactive articles often use is segmenting content into small pieces to be read in-between or alongside other graphics. While we have already discussed cognitive load theory, the Segmenting Theory, the idea that complex lessons are broken into smaller, bit-sized parts [80], also supports personalization within interactive articles. Providing a

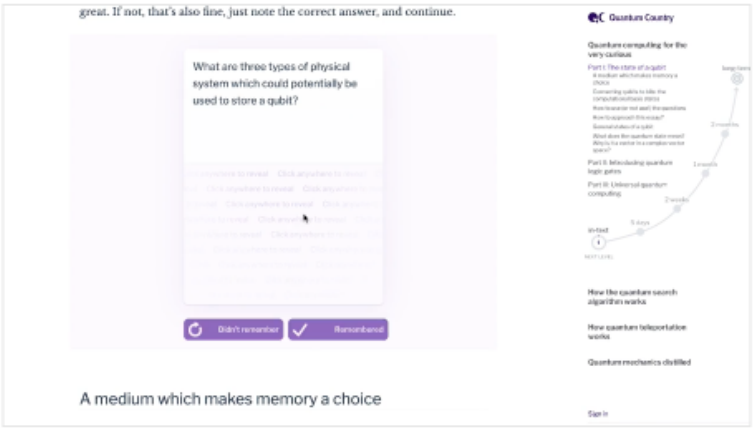


Figure 3.14: In "Quantum Country" [193], the interactive textbook uses spaced repetition and allows a reader to opt-in and save their progress while reading through dense material and mathematical notion over time.

reader the ability to play, pause, and scrub content allows the reader to move at their own speed, comprehending the information at a speed that works best for them. Segmenting also engages a reader's essential processing without overloading their cognitive system [80].

Multiple studies have been conducted showing that learners perform better when information is segmented, whether it be only within an animation [195] or within an interface with textual descriptions [197]. One excellent example of using segmentation and animation to personalize content delivery is "A Visual Introduction to Machine Learning," which introduces fundamental concepts within machine learning in bite-sized pieces, while transforming a single dataset into a trained machine learning model [310]. Extending this idea, in "Quantum Country," an interactive textbook covering quantum computing, the authors implemented a user account system, allowing readers to save their position in the text and consume the content at their own pace [193]. This book further utilizes the interactive medium by utilizing spaced repetition that helps improve recall.

Reducing Cognitive Load

Authors must calibrate the detail at which to discuss ideas and content to their readers expertise and interest to not overload them. When topics become multifaceted and complex, a balance must be struck between a high-level overview of a topic and its lower-level details. One interaction technique used to prevent a cognitive overload within a reader is "details-on-demand."

Details-on-demand has become an ubiquitous design pattern. For example, modern operating systems offer to fetch dictionary definitions when a word is highlighted. When applied to visualization, this technique allows users to select parts of a dataset to be shown in more detail while maintaining a broad overview. This is particularly useful when a change of view is not required, so that users can inspect elements of interest on a point-by-point basis in the context of the whole [94]. Below we highlight areas where details-on-demand has been successfully applied to reduce the amount of information present within an interface at once.

Data Visualization Details-on-demand is core to information visualization, and concludes the seminal Visual Information-Seeking Mantra: *"Overview first, zoom and filter, then details-on-demand"* [264]. Successful visualizations not only provide the base representations and techniques for these three steps, but also bridge the gaps between them [159]. In practice, the solidified standard for details-on-demand in data visualization manifests as a tooltip, typically summoned on a cursor mouseover, that presents extra information in an overlay. Given that datasets often contain multiple attributes, tooltips can show the other attributes that are not currently encoded visually [27], for example, the map in Figure 3.15 that shows where different types of birdsongs where recorded and what they sound like.

Illustration Details-on-demand is also used in illustrations, interactive textbooks, and museum exhibits, where highlighted segments of a figure can be selected to display additional information about the particular segment. For example, in "How does the eye work?" readers can select segments of an anatomical diagram of the human eye to learn more about specific regions, e.g., rods and cones [14]. Another example is "Earth Primer," an interactive

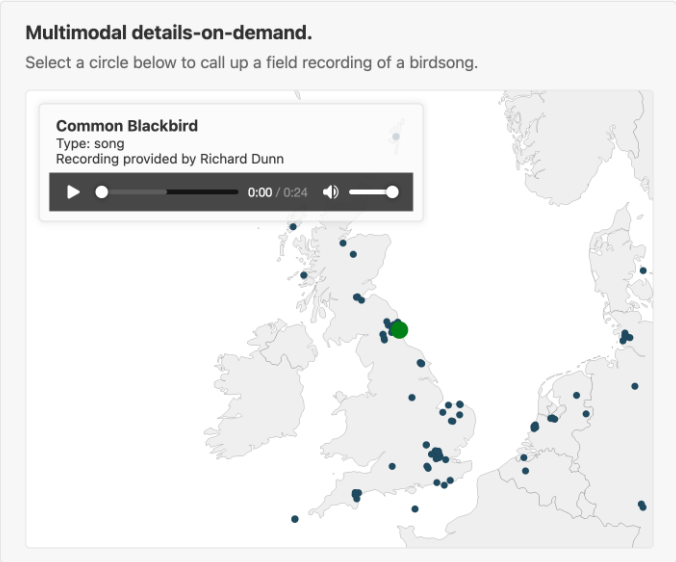


Figure 3.15: In the interactive version of this figure, readers can click any point to listen to a different bird's chirp.

textbook on tablets that allows readers to inspect the Earth's interior, surface, and biomes [119]. Each illustration contains segments the reader can tap to learn and explore in depth. Figure 3.16 demonstrates this by pointing out specific regions in machine-generated imagery [151, 152] to help people spot fake images.

Mathematical Notation Formal mathematics, a historically static medium, can benefit from details-on-demand, for example, to elucidate a reader with intuition about a particular algebraic term, present a geometric interpretation of an equation, or to help a reader retain high-level context while digesting technical details[1]. For example, in "Why Momentum Really Works," equation layout is done using Gestalt principles plus annotation to help a reader easily identify specific terms[120]. In "Colorized Math Equations," the Fourier transform equation is presented in both mathematical notation and plain text, but the two are linked through a mouseover that highlights which term in the equation corresponds to

[1]See `https://github.com/fredhohman/awesome-mathematical-notation-design` for examples that experiment with applying new design techniques to various mathematical notation.

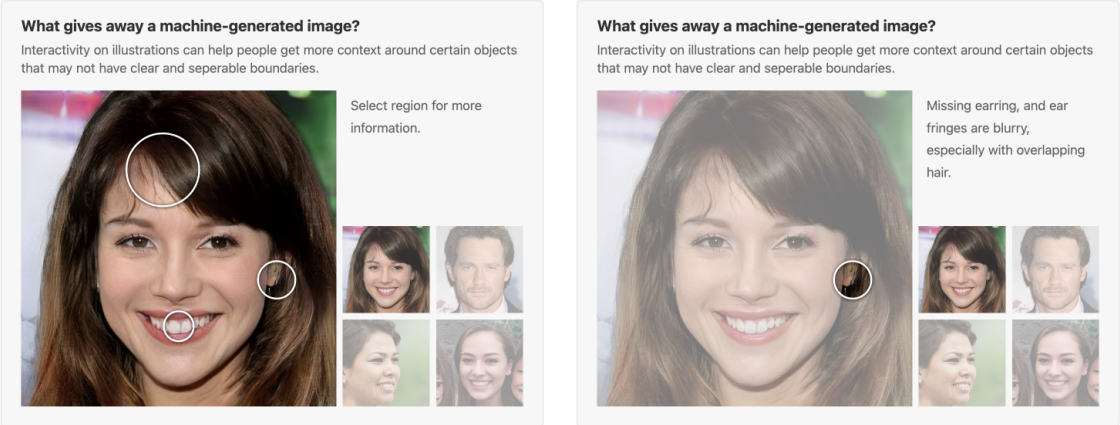


Figure 3.16: In the interactive version of this figure, readers can choose between 1 of 4 machine-generated images and brush over the circle callouts to display a short message about each region. Generated images from [151, 152]

which word in the text [31]. Another example that visualizes mathematics and computation is the "Image Kernels" tutorial where a reader can mouseover a real image and observe the effect and exact computation for applying a filter over the image [235]. Instead of writing down long arithmetic sums, the interface allows readers to quickly see the summation operation's terms and output. In Figure 3.17, one of Maxwell's equations is shown. Click the two buttons to reveal, or remind yourself, what each notation mark and variable represent.

Text While not as pervasive, text documents and other long-form textual mediums have also experimented with letting readers choose a variable level of detail to read. This idea was explored as early as the 1960s in StretchText, a hypertext feature that allows a reader to reveal a more descriptive or exhaustive explanation of something by expanding or contracting the content in place [211]. The idea has resurfaced in more recent examples, including "On Variable Level-of-detail Documents" [46], a PhD thesis turned interactive article [228], and the call for proposals of *The Parametric Press* [16]. One challenge that has limited this technique's adoption is the burden it places on authors to write multiple versions of their content. For example, drag the slider in the interactive version of Figure 3.18 to read

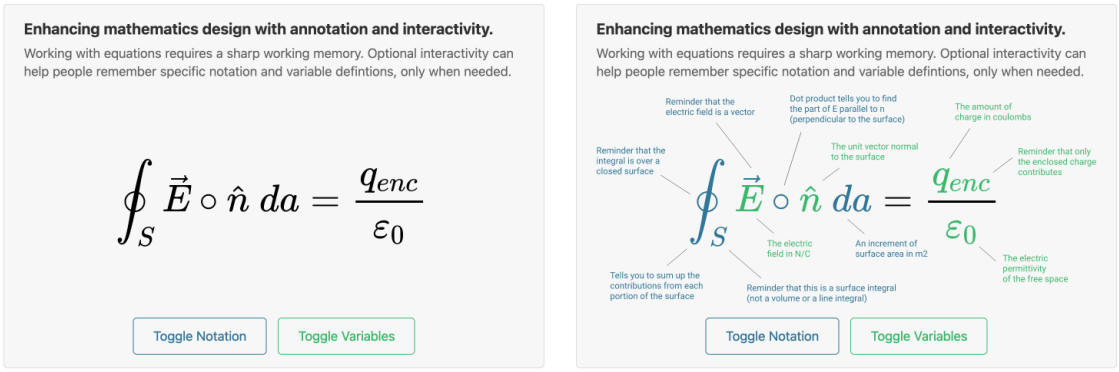


Figure 3.17: In the interactive version of this figure, readers can click to reveal what each mark of notation or variable represents in the equation.

descriptions of the Universal Approximation Theorem in increasing levels of detail. For other examples of details-on-demand for text, such as application in code documentation, see this small collection of examples [41].

Previewing Content Details-on-demand can also be used as a method for previewing content without committing to another interaction or change of view. For example, when hovering over a hyperlink on Wikipedia, a preview card is shown that can contain an image and brief description; this gives readers a quick preview of the topic without clicking through and loading a new page [288]. This idea is also not new: work from human-computer interaction explored fluid links [312, 313] within hypertext that present information about a particular topic in a location that does not obscure the source material. Both older and modern preview techniques use perceptually-based animation and simple tooltips to ensure their interactions are natural and lightweight feeling to readers [312].

3.2 Challenges for Authoring Interactives

If interactive articles provide clear benefits over other mediums for communicating complex ideas, then why aren't they more prevalent?

Unfortunately, creating interactive articles today is difficult. Domain-specific diagrams,

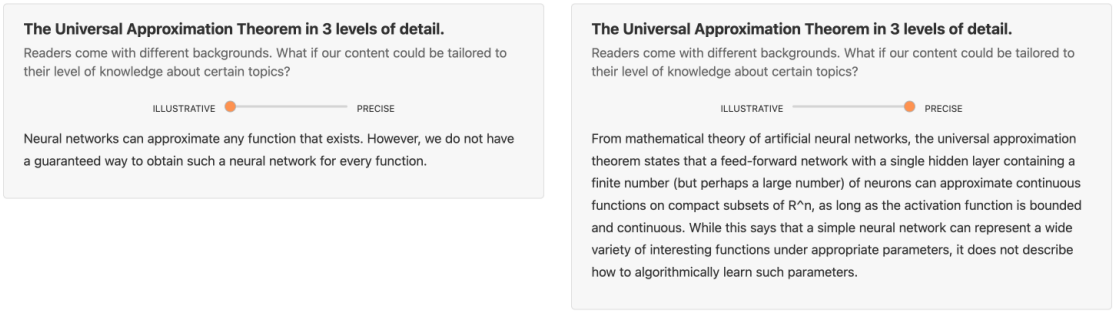


Figure 3.18: In the interactive version of this figure, readers can drag the slider to display the theorem's statement in increasing levels of detail.

the main attraction of many interactive articles, must be individually designed and implemented, often from scratch. Interactions need to be intuitive and performant to achieve a nice reading experience. Needless to say, the text must also be well-written, and, ideally, seamlessly integrated with the graphics.

The act of creating a successful interactive article is closer to building a website than writing a blog post, often taking significantly more time and effort than a static article, or even an academic publication[2]. Most interactive articles are created using general purpose web-development frameworks which, while expressive, can be difficult to work with for authors who are not also web developers. Even for expert web developers, current tools offer lower levels of abstraction than may be desired to prototype and iterate on designs.

While there are some tools that help with alleviating this problem [260, 215, 70, 173], they are relatively immature and mainly help with reducing the necessary programming tedium. Tools like *Idyll* (Chapter 4) can help authors start writing quickly and even enable rapid iteration through various designs (for example, letting an author quickly compare between sequencing content using a "scroller" or "stepper" based layout). However, *Idyll* does not offer any design guidance, help authors think through where interactivity would be most effectively applied, nor highlight how their content could be improved to increase

[2]As a proxy, see the number of commits on an example *Distill* article, e.g., `https://github.com/distillpub/post--building-blocks`.

its readability and memorability. For example, *Idyll* encodes no knowledge of the positive impact of self-explanation, instead it requires authors to be familiar with this research and how to operationalize it.

To design an interactive article successfully requires a diverse set of editorial, design, and programming skills. While some individuals are able to author these articles on their own, many interactive articles are created by a collective team consisting of multiple members with specialized skills, for example, data analysts, scripters, editors, journalists, graphic designers, and typesetters, as outlined in [181]. The current generation of authoring tools do not acknowledge this collaboration. For example, to edit only the text of *this article* requires one to clone its source code using git, install project-specific dependencies using a terminal, and be comfortable editing HTML files. All of this complexity is incidental to task of editing text.

Publishing to the web brings its own challenges: while interactive articles are available to anyone with a browser, they are burdened by rapidly changing web technologies that could break interactive content after just a few years. For this reason, easy and accessible interactive article archival is important for authors to know their work can be confidently preserved indefinitely to support continued readership[3]. Authoring interactive articles also requires designing for a diverse set of devices, for example, ensuring bespoke content can be adapted for desktop and mobile screen sizes with varying connection speeds, since accessing interactive content demands more bandwidth.

There are other non-technical limitations for publishing interactive articles. For example, in non-journalism domains, there is a mis-aligned incentive structure for authoring and publishing interactive content: why should a researcher spend time on an "extra" interactive exposition of their work when they could instead publish more papers, a metric by which their career depends on? While different groups of people seek to maximize their work's impact, legitimizing interactive artifacts requires buy-in from a collective of communities.

Making interactive articles accessible to people with disabilities is an open challenge. The dynamic medium exacerbates this problem compared to traditional static writing, especially

[3]This challenge has been pointed out by the community [224].

when articles combine multiple formats like audio, video, and text. Therefore, ensuring interactive articles are accessible to everyone will require alternative modes of presenting content (e.g. text-to-speech, video captioning, data physicalization, data sonification) and careful interaction design.

It is also important to remember that not everything needs to be interactive. Authors should consider the audience and context of their work when deciding if use of interactivity would be valuable. In the worst case, interactivity may be distracting to readers or the functionality may go unused, the author having wasted their time implementing it. However, even in a domain where the potential communication improvement is incremental[4], at scale (e.g., delivering via the web), interactive articles can still have impact [214].

3.3 Critical Reflections

We write this article not as media theorists, but as practitioners, researchers, and tool builders. While it has never been easier for writers to share their ideas online, current publishing tools largely support only static authoring and do not take full advantage of the fact that the web is a dynamic medium. We want that to change, and we are not alone. Others from the explorable explanations community have identified design patterns that help share complex ideas through play [69, 71, 262, 273, 198].

To explore these ideas further, two of this work's authors created *The Parametric Press:* an annually published digital magazine that showcases the expository power that interactive dynamic media can have when effectively combined [87]. In late 2018, we invited writers to respond to a call for proposals for our first issue focusing on exploring scientific and technological phenomena that stand to shape society at large. We sought to cover topics that would benefit from using the interactive or otherwise dynamic capabilities of the web. Given the challenges of authoring interactive articles, we did not ask authors to submit fully developed pieces. Instead, we accepted idea submissions, and collaborated with the authors over the course of four months to develop the issue, offering technical, design, and editorial assistance collectively to the authors that lacked experience in one of these areas. For

[4]In reality, multimedia studies show large effect sizes for improvement of transfer learning in many cases, see Chapter 12 of [194].

Figure 3.19: The "Myth of the Impartial Machine" [109] was one of five articles published in The Parametric Press. The article used techniques like animation, data visualizations, explanatory diagrams, margin notes, and interactive simulations to explain how biases occur in machine learning systems.

example, we helped a writer implement visualizations, a student frame a cohesive narrative, and a scientist recap history and disseminate to the public. Multiple views from one article are shown in Figure 3.19 [109].

We see *The Parametric Press* as a crucial connection between the often distinct worlds of research and practice. The project serves as a platform through which to operationalize the theories put forth by education, journalism, and HCI researchers. Tools like *Idyll* which are designed in a research setting need to be validated and tested to ensure that they are of practical use; *The Parametric Press* facilitates this by allowing us to study its use in a real-world setting, by authors who are personally motivated to complete their task of constructing a high-quality interactive article, and only have secondary concerns and care about the tooling being used, if at all.

Through *The Parametric Press*, we saw the many challenges of authoring, designing, and publishing first hand, dually as researchers and practitioners. Figure 3.20 summarizes interactive communication opportunities from both research and practice.

As researchers we can treat the project as a series of case studies, where we were observers of the motivation and workflows which were used to craft the stories, from their initial conception to their publication. Motivation to contribute to the project varied by author. Where some authors had personal investment in an issue or dataset they wanted to highlight and raise awareness to broadly, others were drawn towards the medium, recognizing its potential but not having the expertise or support to communicate interactively. We also

	Research	Practice
Authoring	Next generation tooling	Evaluate in production setting, identify bugs
Designing	Developing theory, conducting laboratory studies	Evaluate specific design decisions in the wild, understand constraints
Publishing	Tools, guidelines, and best practices	Concrete examples for others to follow, available source code, accessible archives, DOI, branding

Figure 3.20: Interactive communication opportunities from both research and practice.

observed how research software packages like Apparatus [260], *Idyll* (Chapter 4), and D3 [55] fit into the production of interactive articles, and how authors must combine these disparate tools to create an engaging experience for readers. In one article, "On Particle Physics," an author combined two tools in a way that allowed him to create and embed dynamic graphics directly into his article without writing any code beyond basic markup. One of the creators of Apparatus had not considered this type of integration before, and upon seeing the finished article commented, *"That's fantastic! Reading that article, I had no idea that Apparatus was used. This is a very exciting proof-of-concept for unconventional explorable-explanation workflows."*

We were able to provide editorial guidance to the authors drawing on our knowledge of empirical studies done in the multimedia learning and information visualization communities to recommend graphical structures and page layouts, helping each article's message be communicated most effectively. One of the most exciting outcomes of the project is that we saw authors develop interactive communication skills like any other skill: through continued practice, feedback, and iteration. We also observed the challenges that are inherent in publishing dynamic content on the web and identified the need for improved tooling in this area, specifically around the archiving of interactive articles. Will an article's code still run a year from now? Ten years from now? To address interactive content archival, we set up a system to publish a digital archive of all of our articles at the time that they are first published to the site. At the top of each article on *The Parametric Press* is an archive

link that allows readers to download a WARC (Web ARChive) file that can "played back" without requiring any web infrastructure. While our first iteration of the project relied on ad-hoc solutions to these problems, we hope to show how digital works such as ours can be published confidently knowing that they will be preserved indefinitely.

As practitioners we pushed the boundaries of the current generation of tools designed to support the creation of interactive articles on the web. We found bugs and limitations in *Idyll*, a tool which was originally designed to support the creation of one-off articles that we used as a content management system to power an entire magazine issue. We were forced to write patches and plugins to work around the limitations and achieve our desired publication[5]. We were also forced to craft designs under a more realistic set of constraints than academics usually deal with: when creating a visualization it is not enough to choose the most effective visual encodings, the graphics also had to be aesthetically appealing, adhere to a house style, have minimal impact on page load time and runtime performance, be legible on both mobile and desktop devices, and not be overly burdensome to implement. Any extra hour spent implementing one graphic was an hour that was not spent improving some other part of the issue, such as the clarity of the text, or the overall site design.

There are relatively few outlets that have the skills, technology, and desire to publish interactive articles. From its inception, one of the objectives of the *Parametric Press* is to showcase the new forms of media and publishing that are possible with tools available today, and inspire others to create their own dynamic writings. For example, Omar Shehata, the author of the *Parametric Press* article "Unraveling the JPEG," told us he had wanted to write this interactive article for years yet never had the opportunity, support, or incentive to create it. His article drew wide interest and critical acclaim.

We also wanted to take the opportunity as an independent publication to serve as a concrete example for others to follow, to represent a set of best practices for publishing interactive content. To that end, we made available all of the software that runs the site, including reusable components, custom data visualizations, and the publishing engine itself.

---

[5]Many of these patches have since been merged to *Idyll* itself. This is the power of modular open-source tooling in action.

Chapter 4

IDYLL: A MARKUP LANGUAGE FOR INTERACTIVE ARTICLES

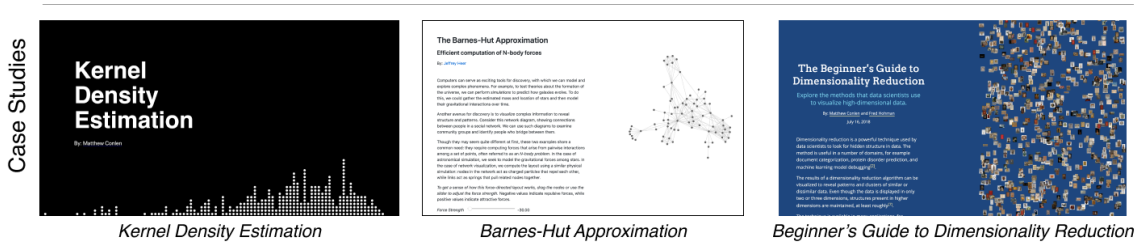


Figure 4.1: *Idyll* been used to create a variety articles—which have been visited over half a million times—including *Beginner's Guide to Dimensionality Reduction* [85], Best Paper Honorable Mention at the Workshop on Visualization for AI Explainability at IEEE VIS.

Publications like the *New York Times*, the *Washington Post*, the *Guardian*, and *FiveThirtyEight* are known for producing high-quality multimedia narratives. Often referred to as *interactives*, these stories have the potential to engage a large audience: in 2013 the most read story in the *New York Times* was an interactive quiz; in 2014, ten of their forty most read stories were from the *Upshot*, a section known for publishing rich data-driven stories. The data visualization research community has attempted to characterize the techniques used in these articles, and have suggested research opportunities in creating tools to drive the production of interactive narratives.

High-quality production pieces are expensive and time consuming to produce. They require custom code, which is often developed by non-expert programmers under the stress of a deadline. Because of this, the resulting web pages can suffer from performance issues (such as being slow to load, especially on mobile browsers), and the code itself may be poorly structured and hard to reuse. In order to improve code quality and decrease development time, some newsrooms create internal frameworks that facilitate common tasks [238, 279,

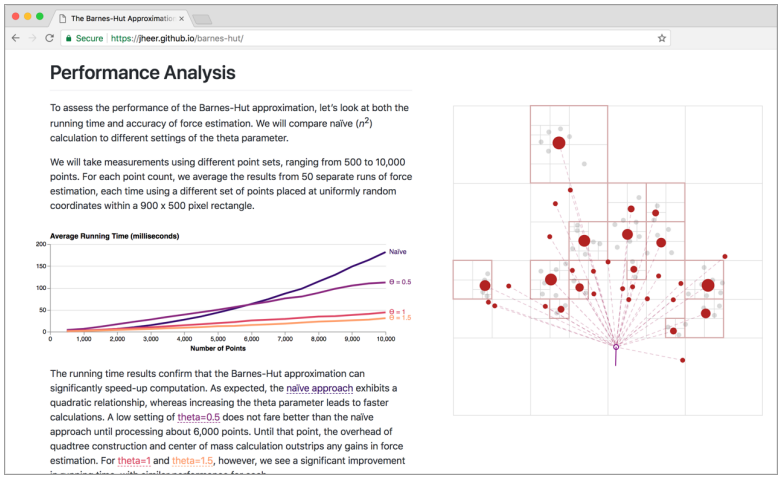


Figure 4.2: An *Idyll* article that interactively links text with visual demonstrations to explain the Barnes-Hut approximation for simulating physical forces. The visualization on the right keeps a fixed position, updating as the user progresses through the article and interacts with the text.

280]. These frameworks, however, require both a large initial effort and maintenance over time, making them inaccessible to all but the most well-staffed organizations.

In this chapter, we contribute *Idyll*, a novel domain specific language (DSL) designed for authoring interactive narratives. *Idyll* targets a balance between expressiveness and readability in its programs, while taking production-quality technical requirements into account. We believe these features make *Idyll* well-suited to provide authors with a way to quickly create and publish interactive articles in a production environment.

Idyll combines a human-readable markup language with a reactive variable system, and provides a straightforward way to embed JavaScript components in-line with text. *Idyll* focuses on making it easy to orchestrate the presentation of the article based on variable state. This allows article behavior to be declaratively specified, removing a large class of code that typically needs to be written when custom interactives are developed, including code that ties HTML elements to JavaScript variables, custom event listeners that trigger page updates, and custom data bindings in the view layer.

Idyll includes a standard library of components that can be used to add interactivity to articles without the need to write JavaScript, and we show how these components cover a wide range of use cases common to interactive articles. *Idyll* exposes high-level language constructs about page state that enable authors to easily parameterize their documents based on scroll events and component properties such as visibility. *Idyll* aims to be easy for non-technical users to pick up, while still being powerful enough to implement a broad range of designs. We show how, on first use, undergraduate students were able to grasp language concepts and produce high quality interactive articles that leverage a broad range of *Idyll* features.

Taken together, the human readable markup, reactive variable system, access to page state & events, library of built-in components, and straightforward extensibility are intended to lower the barrier to entry for creating interactive articles, and accelerate their creation by reducing the overall amount of code and setup needed to produce and publish them. The remainder of the chapter discusses related work, shares our design process, presents *Idyll*, and, through a series of examples and a first use study with students, concretely demonstrates how *Idyll* simplifies the process of creating interactives.

4.1 Related Work

Idyll extends previous work through a novel architecture for reactive parameterization of custom components, high-level bindings to page state and events, and by providing the freedom of customization and interoperability necessary to fit the needs of production-quality work.

Figure 4.3 shows an overview of existing languages and tools for building interactive and data-driven web pages, plotted on the dimensions of *expressiveness* and *readability*. When looking at these dimensions we consider both the threshold and the ceiling [208] that the tools impose for creating interactives. Items below the dotted line on the expressiveness axis have a low ceiling because they do not provide easy ways for users to define custom components, limiting the space of designs that may be produced. Those plotted above that dotted line do allow for users to include arbitrary custom code, and languages that implement features that make it easier to achieve common designs are given higher expressiveness scores.

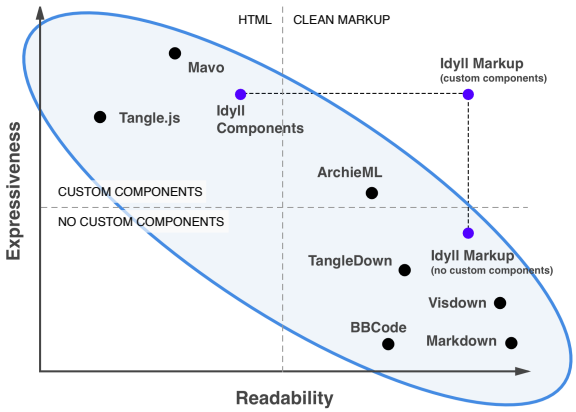


Figure 4.3: Comparing the readability and expressiveness of web-based publishing tools. Many languages focus on providing a legible authoring experience, or on facilitating the implementation of complex designs. *Idyll* aims to achieve high readability of its text, while maintaining expressiveness. It does this by imposing a clean, structured interface between text and interactive elements, providing a standard library, and by allowing users to write custom code where necessary.

For example, both Tangle and ArchieML are used within standard JavaScript, meaning that the authoring environment is powerful because it has all of JavaScript at its disposal.

The readability axis considers an author writing structured text with each language. Many of the items plotted take advantage of a markup language that reduces the amount of code that an author needs to write to style and add hierarchy to their text. Some of the tools require users to write in HTML or JavaScript directly; these tools are shown as having lower readability. When *Idyll* is used without custom components, the markup is relatively simple but expressiveness is limited by the set of standard components; users may write custom JavaScript components, which can be arbitrarily complex. A non-technical user can consider *Idyll* markup without any knowledge of the underlying custom component implementation. From this perspective, the *Idyll markup* will be of a similar complexity regardless of the inclusion of custom components.

4.1.1 Narrative Visualization & Explorable Explainables

Segel & Heer [262] investigated the techniques that are used in data-driven storytelling, looking at stories that have been published by news outlets, and articulated a design space of narrative visualization that has influenced the space of narrative techniques that *Idyll* and other tools attempt to support. In Chapter 2, we noted that one challenge for researchers in this space is that the tools and techniques used by practitioners are constantly shifting, for example both Stolper et al. [273] and McKenna et al. [198] refine Segel & Heer's design space, updating it to reflect changes in practice Researchers have also noted several opportunities for work to be done in this space [177, 181], including evaluating the effectiveness of data-driven storytelling techniques, and building tools that help authors use these techniques in their stories. *Idyll* supports both of these goals, serving as a tool for authoring and publishing data-driven stories, but also serving as a platform for researchers to collect and analyze data about how users interacted with the stories.

4.1.2 Interactive Web Frameworks

Several research systems facilitate end-user creation of data-driven websites, like Gneiss [75], Mavo [289] (see Chapter 2). These tools are aimed at a more general use-case than *Idyll*, but are nonetheless informative when considering programs whose primary *output* are webpages. Flapjax [202] is a language that brings functional reactive programming (FRP) [37] to the web. Flapjax showed that event-driven reactivity is a natural fit for web applications. FRP has been shown to be an effective means for enabling expressive declarative languages in the domains of animation [107] and visualization [259] specification, and can be similar applied in this domain. React [4] is a JavaScript library for declaratively building user interfaces, centered around reactive components. *Idyll* is implemented using the React library, and uses an event-driven reactive parameterization of article specifications.

Other web frameworks utilize a similar component-oriented architecture, for example Polymer [230] is a JavaScript library for building applications using Web Components, a new standard being added to web browsers. These frameworks simplify web developement, however they still require users to navigate complex application code. *Idyll* intentionally

separates JavaScript code and editorial copy, so that non-technical users may still make simple edits to the text of an article without navigating complex code.

4.1.3 Computational Documents

Computational notebook environments such as Jupyter [226] are frequently used by programmers and data scientists to create and share graphics and computations. Observable [215] is a service that hosts computational notebooks written in JavaScript, Like *Idyll*, Observable leverages reactive semantics. However, *Idyll* targets active reading experiences, including layout, navigation, and styling, whereas Jupyter and Observable focus on the coding experience using an interface consisting of a linear list of code and output cells.

Codestrates [239] is a literate computing approach built on web technologies that allows for more customization of the user interface than previous approaches. Where Codestrates allows users to build arbitrary interactive web content, *Idyll* targets a more focused range of design outputs. Other computational document formats include Leisure [65] and Hypercard [29], which also target the creation of more general applications than *Idyll*.

4.1.4 Creating Visualizations & Interactive Graphics

As discussed in Chapter 2, there has been extensive work on tools that facilitate the production of visualizations and interactive graphics. *Idyll* is designed to supplement, rather than supplant, these tools. For example, D3 [55] is a JavaScript library with which developers can create expressive data visualizations and Vega-Lite [257] is a high-level grammar of interactive graphics. *Idyll* considers documents in a more unified and structured manner than D3 or Vega-Lite — an *Idyll* user can easily embed a visualization created with Vega-Lite or D3 in their article.

Compared to existing narrative visualiation authoring tools like Ellipsis [254], which provides a DSL and graphical interface for building narrative visualizations, and TimelineStoryteller [61], a tool for building narrative timelines, Ellipsis, TimelineStoryteller, and *Idyll* supports the orchestration of entire articles, rather than individual components. Other GUI-based tools support the design of specialized graphics. For example, Apparatus [260]

is an editor for creating interactive diagrams; TextAlive [153] is an integrated design environment for creating kinetic typography videos and, like *Idyll*, targets a dual audience of technical and non-technical users. The systems for creating graphics and other media are complementary, as, for example, Ellipsis and TimelineStoryteller could in theory be used to create augmented visualization components or timelines to be included in an *Idyll* article.

4.1.5 Editorial Tools for the Web

There have been many efforts to make it easier to write for the web using a simpler and more concise syntax than HTML. Markdown [125] is a popular markup language designed to be fast to write and easy to read. *Idyll* borrows syntax from Markdown in order to leverage users' familiarity with the language. Jekyll [2] is a blog engine that allows users to write posts in plain text or Markdown files and deploy them to the web, and Visdown [149] extends the idea of compiling Markdown to HTML pages by allowing authors to specify data visualizations in their markup. In contrast with *Idyll*, these tools do not provide a mechanism that allows JavaScript to be tightly integrated with the text. PersaLog [19] is a DSL for dynamic personalization of news articles. *Idyll* similarly provides a DSL for interactive articles, but targets a broader range of components and article designs.

Creating *interactive* documents involves writing custom JavaScript and HTML. It can become difficult to balance the narrative portion of a project with the low-level details of code. To this end, the *New York Times* developed ArchieML [274], a markup language for editors to work with text that will subsequently be used in an interactive article. ArchieML has been adopted by a number of major newsrooms, and is used by both technical staff and non-technical editors. The widespread use of ArchieML shows that editorial staff are able and willing to pick up simple markup languages in order to work more closely with technical collaborators. While ArchieML makes it easy to pull text into code, *Idyll* makes it easy to include JavaScript components in text. We contend that our approach makes the relationship between code and text easier to reason about from an editorial perspective, and enables control over where components appear in text and how they interact with the page. This approach allows *Idyll* to eliminate the need to write a large class of code typically

required when producing these articles.

4.2 Design Requirements

The design of *Idyll*—a markup language for interactive documents—was informed by prior research and motivated by my domain experience working in the digital journalism industry, where cross-platform multimedia and interactive content must be designed, implemented, and published according to a timely newsroom schedule. We aimed to build a tool that would decrease the overall amount of time necessary to create and publish such interactive content, and that would enable a collaborative production process between technical and editorial users. We iteratively refined a design document for the language in response to feedback from domain experts and early use tests, including interviews with professional journalists and designers.

4.2.1 Requirements

The initial draft of the language was driven by knowledge articulated in prior research and obtained by the authors through industry experience. The first author of this paper spent several years authoring interactive graphics and articles, and building tools to support their production and publication across several major news outlets. We derived a set of requirements from our findings regarding a number of social and technical issues.

Process

Lee et al. [181] elucidate the process of producing data-driven stories in their "visual data storytelling process" model. While this model concerns data-driven storytelling specifically, we find that it can be applied more generally to the production of interactive articles: (1) anecdotes, assets, data, and facts are collected; (2) story copy is written, interactive components are designed and scripted; (3) editors refine the content and presentation of the story; (4) the article is published.

This model provides motivation for separating editorial and technical concerns, a need apparent in the design of other tools such as ArchieML. We want *Idyll* to support this

workflow. *Editors should be able to edit copy and arrange interactive widgets without writing code; developers should be able to easily integrate their code into articles.*

Architecture

Real-world processes are more nuanced than abstract models, and specific workflows and requirements may vary across institutions. To support a wide range of workflows, *Idyll should be modular, customizable, and interoperable with existing (and future) tools for creating interactive graphics.* By designing the tool as a set of composable modules, it may either be used holistically or customized to support specific use-cases. By requiring interoperability with other domain-specific tools, we can build a system that is easy to learn on its own, but also supports a wide range of outputs.

In order to facilitate editing of the article by non-technical users, *Idyll* should support *declarative specification of text, behavior, and layout*, rather than requiring imperative code to update document state. The specification format needs to be powerful enough to support interactive behavior; for example it must be possible to implement the type of dynamism shown in Figure 4.4. To achieve this, we draw on functional reactive programming, which has been shown to be an effective means for enabling expressive declarative languages, and can be similar applied in this domain. *Idyll* uses *a reactive parameterization of the article specification.*

Features

While the design space of interactive web articles is a fast-moving target, prior work has attempted to characterize common narrative patterns used in data-driven stories and interactive articles. McKenna et al. [198] present an analysis of navigational techniques that are often used, such as scroll-based navigation ("scrollytelling"), and step-based navigation (e.g., slideshows). In addition to navigation, designers must often implement custom behavior across graphics in response to reader interaction. With currently available tooling, custom code needs to be written to bind variables to widgets and track changes to internal state as readers interact with them. *Idyll* includes *a standard component library to make*

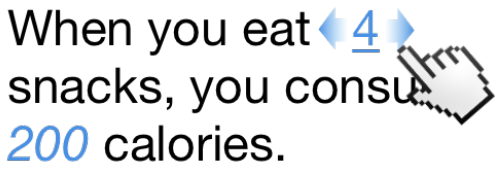


Figure 4.4: A stereotypical behavior of interactive documents: values can be modified by the reader in order to effect change change elsewhere on the page. In this example, adapted from the Tangle's documentation [293], the number of snacks consumed can be modified, and the sentence updates to show the equivalent amount of calories.

common navigation (e.g., scroll- and step-based) and input techniques straightforward to specify.

Publication

Content published online should be fast to load, compatible with a wide range of browsers and devices, accessible to screen-readers, and, in some cases, available in multiple languages. Adhering to the best-practices for serving JavaScript may require a complex build process to transform the code that journalists and developers write into something that can be delivered to a reader. For example, to reduce page load time, developers must limit the number of network requests performed and data transferred. To do this, separate scripts are often compiled into a single file and compressed (e.g., minified). With *Idyll*, we seek to *support the complex JavaScript build configurations necessary for publication.*

Another concern is the way in which HTML is delivered to the reader. In typical web applications an HTML document is rendered by a server, often by querying a database for information and injecting that into an HTML template. To reduce infrastructure costs and deployment complexity, interactive articles typically do not have a database component, and are completely encapsulated by the HTML, CSS, and JavaScript bundle that is delivered to readers. Article content is included directly in the HTML, and JavaScript code attaches event listeners to this existing content when the page is loaded: this process is known as

hydration. Following best practices, *Idyll* should *generate the initial HTML before it is sent to the client, and hydrate interactivity upon page load.*

4.2.2 Interviews

After the initial requirements and candidate language specification were drafted, we solicited feedback from 13 experts from the fields of digital journalism, education, information visualization, and scientific publishing, including journalists and technologists from *FiveThirtyEight*, the *New York Times*, and the *Washington Post*. Twelve of the experts expressed positive reactions and acknowledged seeing value in such a tool (*"Overall the declarative nature of creating the interactive documents here is really neat"*, *"I kinda feel like it's the thing I've always wanted, looking at the [markup] for that page is what convinced me."*). The remaining expert noted that they were *"sometimes puzzled by Markdown specific approaches,"* indicating that they had a hard time seeing how Markdown could integrate into their existing code-heavy workflow.

The experts provided detailed feedback that included suggestions for changes that would improve workflow (*"It might also be worth supporting comments in the syntax. In our ArchieML documents, we've used comments occasionally to make suggestions and to explain the syntax a bit to copy editors"*), requests for additional features (*"It would be a nice feature to have a csv to json converter as part of the build"*), and notes about potential technical hurdles (*"Async stuff is always... interesting in this sort of project, e.g. if you need to make chained network requests and do something with the end result"*). We incorporated this feedback into our requirements document and built the initial version of *Idyll*.

4.3 The Idyll Language

Idyll is a markup language that targets interactive browser-based articles, generating HTML, JavaScript, and CSS. An *Idyll* document can contain both *textual markup* using Markdown syntax and *component markup* used to declare variables and interactive components. Listing 4.1 shows basic usage of *Idyll* syntax. To produce the output, the *Idyll* compiler takes in a markup file and creates a list of article components. Figure 4.5 shows how these components are combined with style information in order to produce a webpage that may be

published online. *Idyll* implements a reactive runtime that responds to user input events and updates rendered output in response.

4.3.1 Language Constructs

The basic language primitives in *Idyll* consist of text, components, reactive variables, and style directives.

Text

The most basic building block of an *Idyll* article is text. Users can write plain text in *Idyll* markup and this text will appear in the rendered output. Text can be written using Markdown syntax, allowing users to easily create lists, headers, blockquotes, and stylized text (e.g., bold or italicized). Markdown enjoys widespread use, with a syntax understood by a wide audience that includes non-technical writers and editors.

Components

Idyll extends Markdown with support for embedded interactive components and variables. The components may either refer to a standard *Idyll* component, a third-party component, or a custom component provided by the author. Components are specified by a name and parameterized by a list of properties, the values of which can be dynamic. When a variable is declared it is added to a global state object that listens for changes to the variable; upon changing, *Idyll* will re-render any components which depend on the value of that variable. *Idyll* must find an implementation for each component included in the markup: to do this it matches the name against author-provided custom components, a standard library of components, installed third-party components, and—if no matches are found—valid HTML tags.

Idyll's standard library of components consists of 26 components, chosen to help authors implement a range of common design goals without writing JavaScript code. While we don't expect *Idyll* to eliminate the need to write code, the standard library serves to limit the code that users do need to write for their specific content and story.

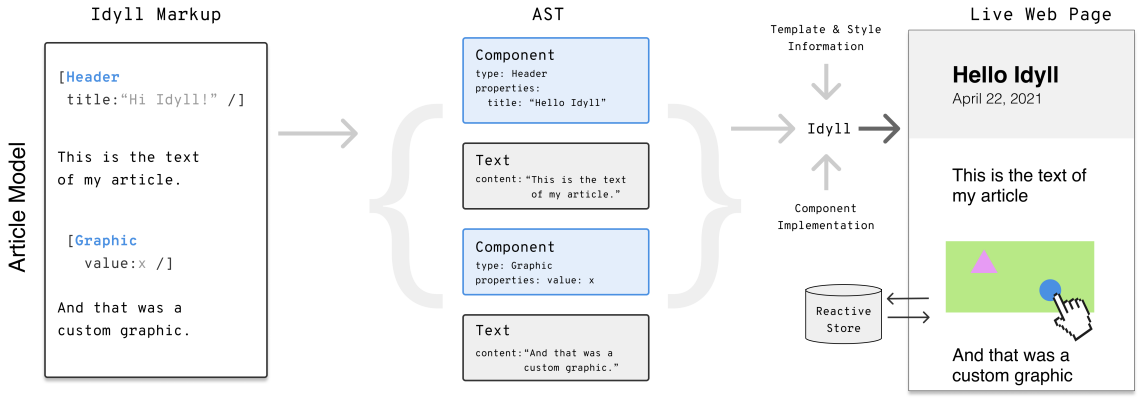


Figure 4.5: The *Idyll* article model. (1) The *Idyll* compiler transforms input markup into a list of document nodes; (2) each of these nodes has a dictionary of properties that determine its behavior (optionally including a list of children which are recursively rendered). (3) The nodes are combined with theme and layout information to (4) construct a static HTML page. When the page is loaded in a browser it is (5) hydrated with event handlers and a reactive state to drive interface updates.

```
# I am a header

Variables can be declared anywhere:
[var name:"x" value:5 /]

and *Markdown* syntax can be used
for inline styling.

The value of x is [Display value:x format:"d" /].

[Range value:x min:0 max:10 /]
```

Listing 4.1: Example *Idyll* markup: Markdown syntax is extended to support variable declaration and interactive component specification. This input file compiles to an HTML, JS, and CSS bundle that can be rendered by all major web browsers.

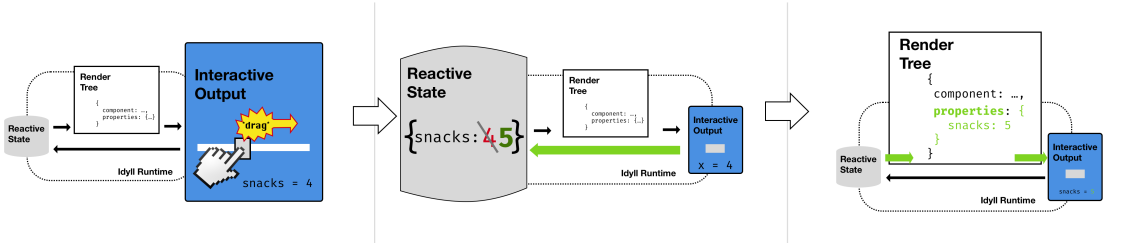


Figure 4.6: In *Idyll*, a document's variable values can be bound to control widgets, allowing authors to quickly implement components that respond to user input. This graphic shows how reader interaction propagates through *Idyll*'s state and affects the rendered output.

The standard library components span four categories:

Presentation components display content on the screen, for example the `Equation` component renders typeset LaTeX code, and the `Table` component renders tabular data.

Input components accept input from users to drive the behavior of articles. For example, the `Range` component adds a slider that can be bound to a variable; the `Select` component displays a list of items in a dropdown menu.

Layout components manipulate how content is displayed. These components can be used as building blocks to create many common designs seen in the narrative visualization space. For example, the `Fixed` component defines content that stays fixed to the screen while a reader scrolls through an article. The second and third examples in the Examples section demonstrate the use of several of these components.

Meta components help with behind-the-scenes tasks and do not modify visible content. Meta components may be used to add document metadata or usage analytics to an article.

Reactive Variables & State

Idyll maintains an internal state that determines how the components are rendered at any point in time. The state consists of user-declared variables and additional information about components with respect to the reader's viewport, for example whether or not a particular component is in view. Variables are declared using a similar syntax to components

(Listing 4.1).

Idyll's variables are reactive: any time the value of a variable is modified, dependent components are re-rendered to reflect the change. Figure 4.6 shows how this works in practice. The article is rendered as a function of the input content and initial state; an event occurs that causes the state to be updated, for example a "click" event occurs when the user clicks a button; the updated state propagates to the article's components, and the webpage is re-rendered to reflect the changes.

Variables are modified in response to events. *Idyll* exposes all events provided by the browser's Document Object Model (DOM), including clicks (and taps), mouse hover, and keyboard input. It also adds several high-level viewport events that are useful for defining scroll-driven interactions. For example, the following code specifies that a custom data visualization will only animate while it is fully in a reader's viewport:

```
[var name:"isAnimating" value:false /]

[CustomDataVisualization
  animating:isAnimating
  onEnterView:`isAnimating = true`
  onExitViewFully:`isAnimating = false` /]
```

Variables may also be updated by components: a special function `updateProps` is provided, which can be called to tell *Idyll* to update the value of a variable which is bound to a particular property of that component. For example, in Listing 4.1 the `Range` component's property `value` is bound to the variable `x`. When the user interacts with the rendered range slider, the Range component calls `updateProps({ value: newValue })`, and *Idyll* subsequently updates the value of `x`.

Style

In web programming, styling of content typically is done using CSS [183], which can be used to specify colors and typography as well as element sizing and layout. *Idyll* adds additional structure to how articles are styled: the look and feel of an article is determined by its *layout*, *theme*, and any *custom styles* provided by the article author. The layout is

responsible for the overall page structure, specifying, for example, if the text container is centered on the page or left aligned, and where the article title should appear. The theme is responsible for colors, typography, and component-specific styles. The separation of form and content offers authors an easy way to view their articles under a variety of different stylings without making code changes, while also offering a way for institutional styles to be applied across an organization's articles without requiring authors to adapt their workflow.

Idyll includes two layouts and three themes, which are suitable for a range of use cases. The *blog* layout utilizes left aligned body text with a wide right margin to accommodate the positioning of complex graphics; the *centered* layout is a more traditional center aligned article layout. The three themes *github*, *tufte*, and *idyll* are different skins inspired by the style of GitHub READMEs, the print design of Edward Tufte, and the *Idyll* documentation site's visual design, respectively. Users can extend and modify these themes using CSS, and can also install themes that others have shared.

4.3.2 Implementation

Idyll is implemented via a collection of composable modules. A command-line tool is used to compile source files and generate output that can be viewed in a web browser. This output contains a JavaScript runtime that reactively updates article components in response to reader actions. These modules are all available on GitHub as open source software.

Build

Projects are compiled using a node.js-based command-line tool. Users specify an *Idyll* markup file as input, along with any custom interactive components (written in JavaScript) and datasets. Users can also optionally define custom article themes and layouts to control the look and feel of the resulting output (e.g., using CSS). The input markup is transformed by a compiler into an abstract syntax tree (AST). The nodes in the AST are matched with interactive components which are then sent through a JavaScript bundler and minifier, resulting in a production-ready code package, which can be deployed to any static web host.

Runtime

The *Idyll* runtime renders interactive output according to the bundled AST and JavaScript components. The runtime provides a reactive variable store implemented using React.js [4], a reactive web framework. *Idyll* is compatible with any preexisting React components; *Idyll* users have access to an existing repository of thousands of open source components.

4.4 Examples

We present a number of example *Idyll* programs to substantiate our claims that *Idyll* (1) reduces the amount of code and effort needed to create interactives, (2) promotes clear and concise code, and (3) is able to express a wide range of designs. A variety of full examples have been produced with *Idyll*, available for viewing both online. To open these examples in a browser, visit `http://idyll-lang.org/`.

The examples that follow are based on techniques used in real-world articles, and focus on explaining how specific design techniques can be achieved concisely using *Idyll* markup. They are illustrative of the way in which *Idyll*'s architecture and language features eliminate the need for much of the code that goes into powering interactive articles, and demonstrate the language's expressiveness. We encourage readers to view the examples in a web browser to gain a more intuitive understanding of the types of designs and interactivity supported.

4.4.1 Reactive Updates to User Input

A common design in the domain of interactive text is to embed text and numbers that change based on a reader's input. The following is an example taken from Tangle's documentation, recreated using *Idyll*. Figure 4.4 shows possible output. The example displays the sentence "When you eat 4 snacks, you consume 200 calories." The number `4` is dynamic: a reader may change its value with a drag interaction, in turn updating the displayed number of snacks and calories consumed.

The *Idyll* implementation of this program is shown in Listing 4.5. The program starts by declaring the variable `snacks` and initializing its value to 4. The following text renders a sentence with the form "When you eat _____ snacks, you consume _____ calories", using

```
[var name:"step" value:0
    /]

[Stepper fullWidth:true
    currentStep:step]
  [Graphic]
    [CustomComponent state
        :scrollStep /]
  [/Graphic]

  [TextContainer]
    [Step]
      Text for the first
          section
    [/Step]
    [Step]
      Text for the second
          section
    [/Step]
    [Step]
      Text for the third
          section
    [/Step]
  [/TextContainer]
[/Stepper]
```

Listing 4.2: *Idyll* markup for stepper-based navigation, in which a user clicks through slides of content. At each stage the `step` variable is updated, causing the custom component to update its state.

```
[var name:"step" value:0 /]

[Scroller fullWidth:true
    currentStep:step]
  [Graphic]
    [CustomComponent state:
        step /]
  [/Graphic]

  [TextContainer]
    [Step]
      Text for the first
          section
    [/Step]
    [Step]
      Text for the second
          section
    [/Step]
    [Step]
      Text for the third
          section
    [/Step]
  [/TextContainer]
[/Scroller]
```

Listing 4.3: *Idyll* markup for scroll-based navigation, in which a graphic will stick in the viewport when the user is scrolling through each step, and release otherwise. The `step` variable updates as each step comes into view.

```
[var name:"state" value:"
    initial-state" /]

[Fixed fullWidth:true]
  [CustomD3Component state
      :step /]
[/Fixed]

[Scroller currentState:
    step]
  [TextContainer]
    [Step state:"initial-
        state"]
      Text for the first
          section
    [/Step]
    [Step]
      This step will not
          trigger a state
          update
    [/Step]
    [Step state:"secondary
        -state"]
      Text for another
          section
    [/Step]
  [/TextContainer]
[/Scroller]
```

Listing 4.4: *Idyll* markup for the layout seen in Figure 4.7. A graphic stays fixed fully in the background while the reader proceeds through the article. The scroller is used to send state updates to the graphic based on the reader's progress.

```
[var name:"snacks" value:4 /]

When you eat [Dynamic value:snacks /]
snacks, you consume
[Display value:‘50 * snacks‘ /] calories.
```

Listing 4.5: This *Idyll* markup produces the behavior shown in Figure 4.4, displaying a dynamic number which when changed will cause later text in the sentence to update.

`Dynamic` and `Display` components to fill in the blanks. The `Dynamic` component creates a two-way binding between the `snacks` variable and the number rendered on-screen. The number is rendered with visual cues showing that it can be dragged, and if the user modifies it, the new variable value is propagated to dependent components.

Idyll allows expressions to be passed directly as properties, eliminating the need to create a separate derived variable in many cases. The `Display` component renders the value of an expression, $50 * snacks$. As the user interacts with the `snacks` variable, the value displayed is updated. If another variable is needed for clarity, *Idyll*'s `derived` variables can reactively compute a value based on other variables.

4.4.2 Fixed Sections and Scroll Events

Interactive articles often dynamically update on-screen content as a reader scrolls through the page, modifying the layout in more complex ways than in the previous example. For example, it is common to have a graphic scroll into view alongside text, and stay fixed in view until the reader scrolls through to the next section. To facilitate this type of interaction, *Idyll* includes a `[Scroller /]` layout component that lets authors declaratively define content that will stay fixed in their readers viewports' while they scroll. Listing 4.3 shows the *Idyll* markup necessary to construct such a scrolling experience. Multiple `[Scroller /]` components can be listed to define a series of scroll sections and their associated graphics.

Figure 4.7 shows an example of an *Idyll* article that uses this technique. The article,

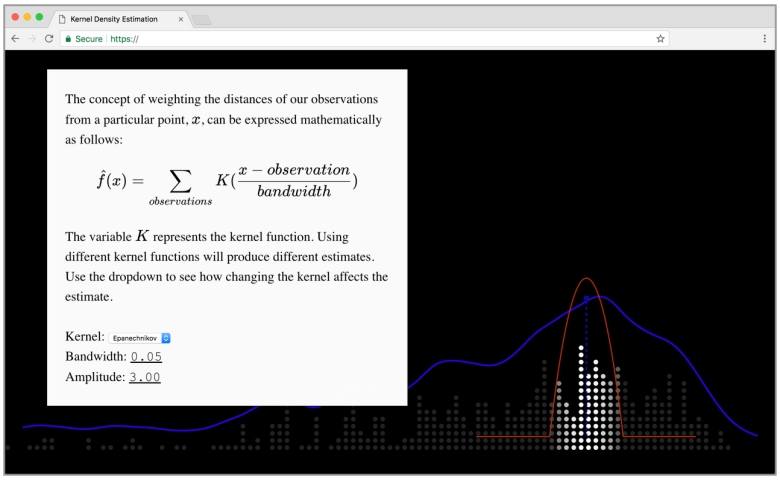


Figure 4.7: An interactive article explaining Kernel Density Estimation. The article uses scroll-based interactivity defined declaratively in *Idyll* markup. Listing 4.4 shows the *Idyll* markup used to specify this layout. The equations and controls shown in this screenshot were defined in *Idyll* markup and reactively parameterize the custom graphic.

which interactively explains kernel density estimation (KDE), displays a fullscreen fixed graphic that stays in a reader's viewport while they scroll through the article. As they reach certain sections, waypoints are triggered which update *Idyll* variables, this update in turn causes the graphic to render in a new state.

One of the benefits of declaratively specifying article components in this way is that authors can quickly explore alternative designs. For example, Listing 4.2 shows the code for displaying the same graphic using a *stepper* design[1]. Note the similarities between Listing 4.3 and Listing 4.2: both define a graphic component and a series of steps and both use a variable to track the reader's progression through the content.

Both the Scroller and Stepper components accept properties that allow authors to further customize their behavior. For example, an author can use the Scroller component with the `disableScroll` property to specify the hybrid scroller-stepper navigation characterized by McKenna et al. [198]. This type of navigation, which can be seen in popular data-driven

[1]A *stepper* can be thought of as a generalization of a slideshow. The graphic shows the content of only one step at any given time, and is updated when the `currentStep` property changes.

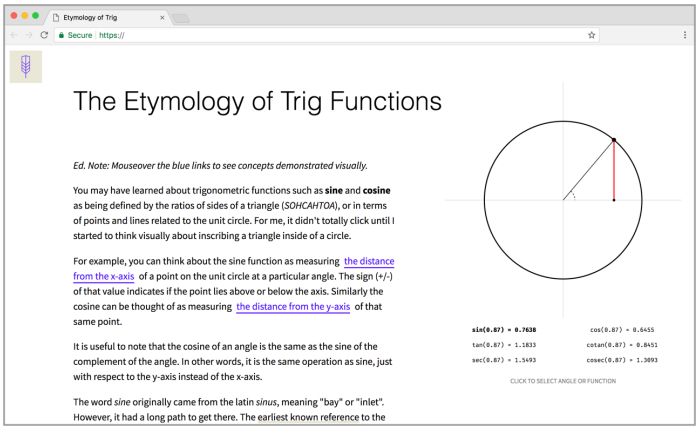


Figure 4.8: *The Etymology of Trig Functions*, an interactive blog post written with Idyll. This example integrates narrative and graphics through user interaction with the text. As a user hovers their mouse over stylized text, the graphic on the right updates to show a geometric interpretation of the concept being discussed.

stories such as *Rock n' Poll* by Maarten Lambrechts [178], overrides the browser's default scroll behavior, requiring readers to click to scroll to the next section.

4.4.3 Updating a Custom Visualization

The previous example showed how *Idyll*'s responsive variable system and built-in components can be used together to create interactive experiences without writing custom JavaScript. However, in many real-world scenarios authors will need to write code to create custom components for their articles. In this example we show how responsive variables can be used to parameterize a custom visualization built with D3. The interface exposed by *Idyll* enforces separation of concerns between editorial issues (the text of the document, order of sections, events that trigger certain content to be displayed, etc.) and specific component implementation details.

Figure 4.8 shows a screenshot of *The Etymology of Trig Functions*, an interactive blog post made with *Idyll*. The post uses a custom visualization in order to illustrate the history of some trigonometric terms. This story uses interactive text components to invite the

```
[var name:"state" value:"init" /]

[Action onMouseEnter:`state = "showCosine"` ]
  Mouse over this text to see the "cosine" state.
[/Action]

[Fixed position:"right"]
  [TrigDisplay state:state /]
[/Fixed]
```

Listing 4.6: *Idyll* markup similar to that used by *The Etymology of Trig Functions*, shown above. As a user hovers their mouse over the action component (which renders stylized text), the state variable will update, causing the fixed graphic to update.

```
initialize(node, props) {
  // render initial graphics
  this.svg = d3.select(node).append('...')
}

update(props, oldProps) {
  // update based on new page state
  this.svg.selectAll('...')
}
```

Listing 4.7: JavaScript interface for custom components. A custom *Idyll* component requires implementing two functions: `initialize` and `update`. The initialize function is called only once, on page load, and is used to render the initial view of the graphic. The update function is called when a component's properties change, for example in response to a user moving a slider or scrolling to a new section.

reader to interact with the graphic as they progress through the text. The visualization updates as readers interact with different components on the the page.

Listing 4.6 sketches how the article in Figure 4.8 can be created. The `[Action /]` tag displays text that can respond to user events, in this case `onMouseEnter`, which fires when

a reader hovers their mouse over the text. The `Action` component is used in conjunction with the `[Fixed /]` component, which causes its contents to remained fixed on-screen in the article's margin as the reader scrolls. We also include `TrigDisplay`, a tag which instructs *Idyll* to load a custom JavaScript component in a file named `trig-display.js`.

The skeleton of the custom `TrigDisplay` component code is shown in Listing 4.7. A component author must implement `initialize` and `update` functions. The `initialize` function is called when the component is first added to the page. The `update` function is called every time a relevant *Idyll* variable updates, and accepts the new and previous values of the component properties as input.

These same *Idyll* mechanisms — `Action` links, `Fixed` layout, and custom components — were used to author the article shown in Figure 4.2, which uses an interactive data visualization to explain the Barnes-Hut approximation of n-body forces. Hyperlinks and input components embedded in the text parameterize the main visualization state, while users can also interact directly with the visualization to explore on their own.

4.5 Deployment

Idyll has been released as free and open source software, and has been downloaded over 25,000 times [6] since its initial release. The code is available at `https://github.com/idyll-lang/idyll`, and the language is available for testing in an online editor at `https://idyll-lang.org/editor/`. The project has received positive feedback from a number of professional journalists and software developers. When asked about an early version of *Idyll*, one *Washington Post* reporter with JavaScript experience noted, *"I finally played around with Idyll a bit and it seems great. [I] got everything working with no problem and the syntax was super easy to use. I didn't dig too much into building something crazy, but I still was able to create a couple custom components and charts."*

Idyll has been well received by the open source programming community, and has received substantial core code updates from outside contributors: five external contributors have made 280 code commits to *Idyll*'s core modules, and others have helped improve the documentation and examples. Community members have used the language to make their own examples and posts. One graphics programmer uses *Idyll* to power his personal blog,

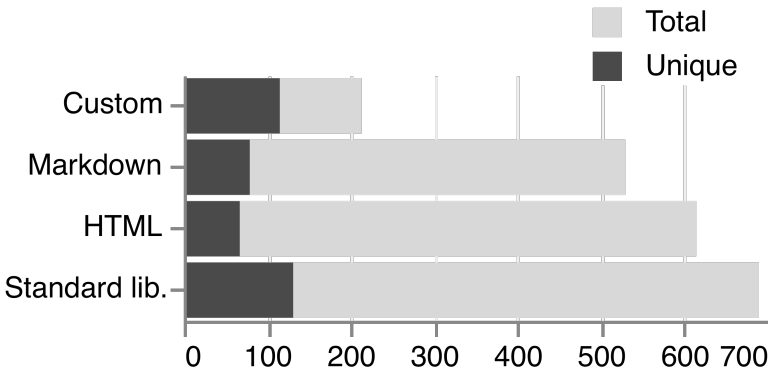


Figure 4.9: Usage counts of various language features across student groups, along with the total unique uses of features per group. The students were comfortable using the Markdown syntax, and relied heavily on *Idyll*'s standard library of components, reducing the overall amount of code that they had to write.

having converting it from an existing site powered by Markdown. He commented, *"Sharing components and styles seems to be working really well! . . . Very pleasant experience overall. Life gets way simpler when you quit trying to be fancy and just make it work."*

After releasing the initial version of *Idyll*, the authors worked with Folo Media, a non-profit news outlet, to produce a data-driven story about school funding in Texas [2]. While the story was never published for reasons unrelated to *Idyll*, the process was illustrative of how the tool can be used in practice. The story was written by a journalist (who was familiar with Markdown and had basic experience with the programming language R, but was otherwise non-technical), was edited by several non-technical staff members at Folo Media, and required the creation of several custom components, which were programmed by the first author. The journalist was able to pick up *Idyll* markup basics without issue and saw the value of the workflow that *Idyll* enabled, although he hoped for a more streamlined way to incorporate charts created during data analysis in R. *"Is anyone working on an R wrapper for Idyll? I feel like your language is like the ideal way for developing interactive narratives. I know folks have been working on ways to better integrate JavaScript into the*

[2] A draft is available at `https://mathisonian.github.io/texas-school-finance/`.

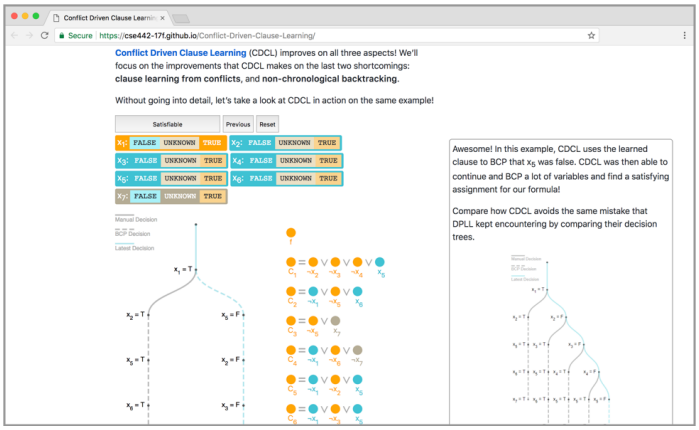


Figure 4.10: A student-authored *Idyll* article explaining the conflict-driven clause learning (CDCL) SAT solver. The article displays custom visualizations parameterized by *Idyll* variables and standard components. The detail shown above visualizes a logical formula; readers can use the buttons to toggle the truth assignment of variables, and see the effects on clauses and the overall formula.

R universe." The editors did not directly engage with the *Idyll* markup, instead looking at drafts of the article in a web browser and sending targeted notes via chat and email, but they were able to participate in an iterative design dialog enabled by *Idyll*'s standard component library (*"What if we made this component a Stepper instead of a Scroller?"*) and decided to add additional content (audio interviews) to the article after realizing *Idyll*'s support for such rich media.

4.5.1 Idyll in the Classroom

An undergraduate data visualization class used *Idyll* for their final projects. The computer science students were required to form groups of four and construct an interactive article that explained an algorithm of their choice. While students are not the intended end users of *Idyll*, they represent a reasonable proxy for the technical users that we wish to engage. The students are technically proficient but far from expert web developers; the study does not speak to usage by non-technical editorial users. Of the 20 groups, 19 of them successfully

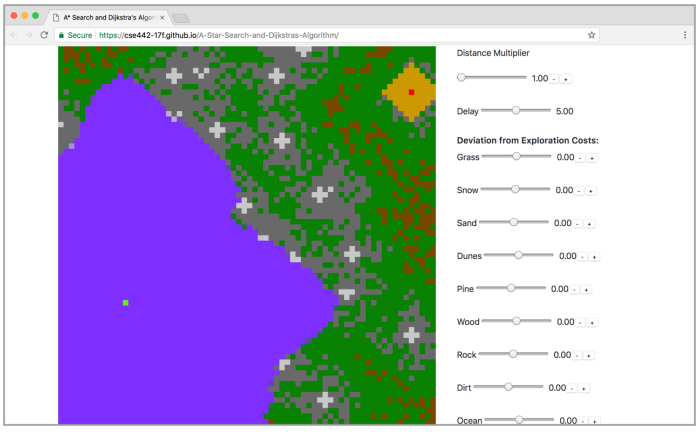


Figure 4.11: This student article, *A* Search and Dijkstra's Algorithm*, is motivated by the use of path-finding algorithms in video games. The students developed a custom game, controlled via *Idyll* components, in which players choose optimal search parameters in order to win.

used *Idyll* to create an interactive article. (One group decided to focus on making video graphics and opted to embed these videos directly in HTML.)

Figure 4.9 shows how the students utilized different language features in their articles. All student groups were comfortable using Markdown, and all took advantage of *Idyll*'s built-in components. The high use of *Idyll*'s standard components indicates that our standard library reduces the overall amount of code that authors need to write, as they don't need to implement a range of functionality themselves. We found some of the student usage of HTML tags to be due to students needing to work around issues in *Idyll*'s compiler, (for example, many groups used the `[br /]` tag to insert extra whitespace where the compiler had stripped it away), though they also used semantic tags such as `[section]` `[/section]` to group their content.

All but one of the groups included their own custom components using *Idyll*'s JavaScript bundling infrastructure. The group that did not include any custom *Idyll* components wrote their JavaScript separately and included it in the article via `[iframe /]` tags. The majority of the custom components that were developed were algorithm-specific visualization

components. The basic components provided by *Idyll* covered a wide range of use cases for capturing reader input and describing document layout and scroll behavior, allowing students to focus on writing code for custom algorithm visualizations.

Figure 4.10 presents an article about *conflict driven clause learning* (CDCL), an algorithm for solving the boolean satisfiability problem. Using the game Sudoku as motivation, the student authors walk readers through the steps of CDCL and related algorithms. The piece concludes with an interactive Sudoku solver that compares CDCL with another solver. The students were able to combine standard *Idyll* components with custom visualizations to create an engaging visual explanation.

Teams were able to use *Idyll* to create sophisticated interactive experiences. Figure 4.11 shows a screenshot from an article about A* search and Dijkstra's pathfinding algorithm. The narrative discusses the importance of path-finding algorithms in video games, and uses interactive graphics to display how the algorithms would find paths to varying points in game levels. The students included an interactive game, *aStarCommando*, where players tweak parameters of a search algorithm in order to navigate to a safe house located across the map before they are killed or captured. The game was parameterized by *Idyll*'s reactive variables and controlled by *Idyll* input widgets.

Some students took advantage of *Idyll*'s layout components and flexibility with styling. Figure 4.12 shows an article on the classic Travelling Salesman Problem (TSP). The student authors of this article customized the color theme, typography, and standard component styles. They used Idyll's variables and components to power interactive maps, upon which different TSP algorithms are demonstrated. The students used a combination of scroll-based and step-based navigation to produce an article with a sophisticated look and feel.

While students had few issues picking up the basic syntax and workflow, some faced initial difficulties adjusting to the declarative, reactive nature of *Idyll* markup. Rather than coordinate interaction via *Idyll* variables and component properties, some students attempted to code coordination themselves by setting and reading global variables in JavaScript. Others sought to invoke methods defined on custom components directly from the markup.

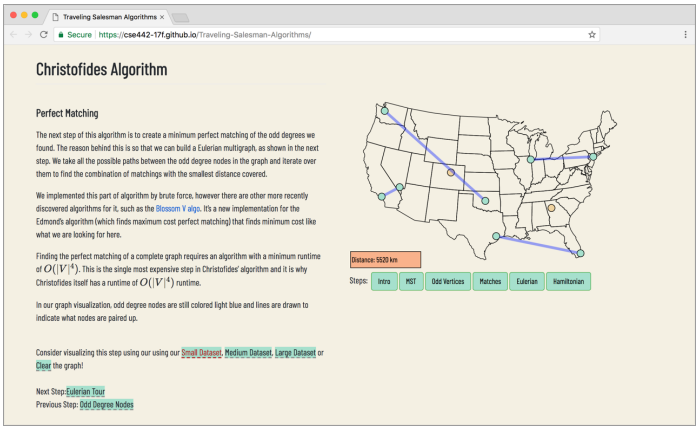


Figure 4.12: A student article explaining the Travelling Salesman Problem. The students leveraged *Idyll*'s features for modifying styles and layout. They were able to create a highly personalized page, incorporating both scroll- and step-based navigation.

4.6 Discussion

4.6.1 Limitations

While the results of the classroom study indicate that technical users are able to use *Idyll* to create a range of designs, the study does not speak to usage by non-technical editorial users. We believe that the widespread adoption of ArchieML, along with the feedback we've received from journalists, show that these users are willing and able to adopt simple markup languages. A more targeted study is needed to say how usable *Idyll* is for editorial users in its current form, and what additional training, if any, would be necessary for them to use the tool effectively.

Idyll does not eliminate the need for writing JavaScript code to create custom graphics. Custom *Idyll* components can grow to become arbitrarily complex, and beyond providing reactivity, *Idyll* doesn't do much to decrease the complexity of these custom graphics. A closer integration with tools that focus on the creation of custom graphics may be needed in order to lower the threshold for creating articles that include custom components. *Idyll* might also benefit from a closer integration with other existing data science tools, as many

data journalists use tools like R and Python to create static graphics. A more streamlined workflow might, for example, allow a user to embed R code directly in their markup and have a static graphic be generated at compile time.

Idyll's two-way variable binding allows components to trigger page updates by modifying *Idyll* variables via a special function, however two-way binding to derived variables is not supported. This distinction may be confusing to some users, and it may be addressed in a future version by adding a constraint solver to *Idyll*'s runtime.

4.6.2 Conclusion

We contribute *Idyll*, a novel domain-specific language for creating interactive narratives. *Idyll* combines a simple markup language with reactive JavaScript components to reduce the amount of code and effort needed to produce and publish interactive articles. Code, examples, and documentation are available online at `https://idyll-lang.org/`.

Chapter 5

IDYLL STUDIO: STRUCTURED EDITING OF INTERACTIVE ARTICLES

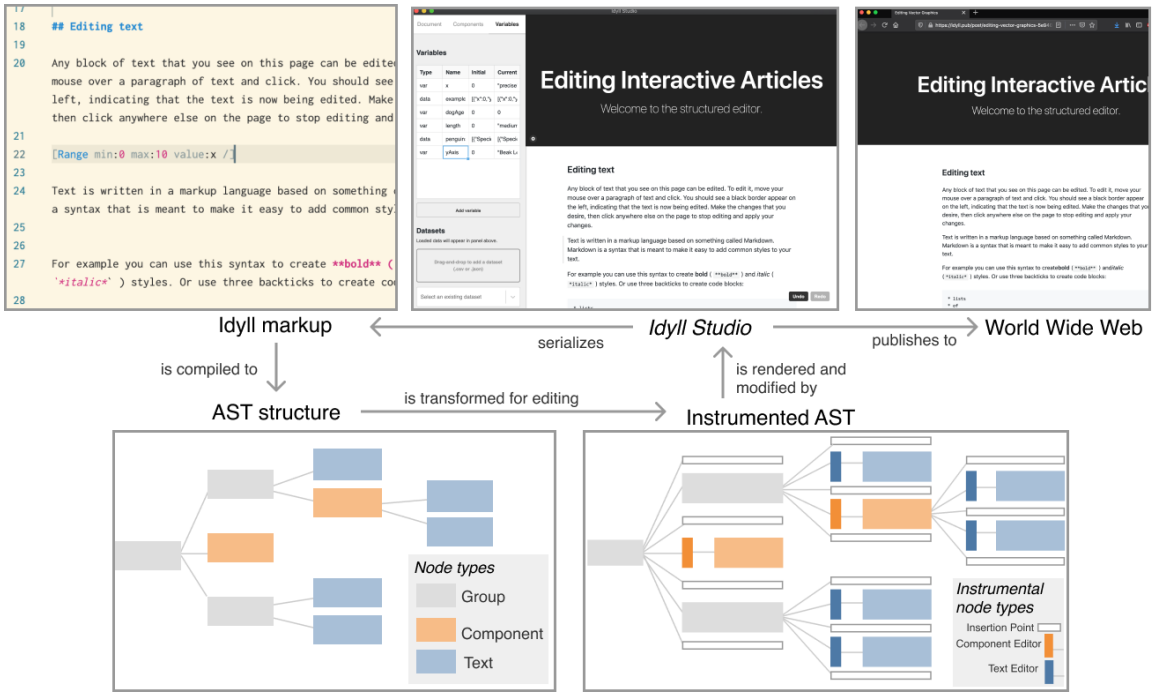


Figure 5.1: *Idyll Studio* builds on the *Idyll* markup language by creating a structured editing interface for creating, manipulating, and publishing interactive articles. We implement a tree expansion algorithm to transform arbitrary *Idyll* programs into structured editing interfaces. *Text editors* allow an author to edit and style text while *component editors* allow for orchestration of component behaviors. *Insertion points* can be used to add new components or text blocks to the article via drag-and-drop.

Interactive articles like Victor's *Explorable Explanations* [291] utilize dynamic text, manipulable controls, and interactive graphics to present information to viewers in an engaging,

reader-driven form. This format is a useful medium of communication across domains like journalism [261, 124], education [194], and scientific publishing [102] because interactive articles can promote active reading [291], foster engagement [124], and lead to improved learning outcomes [167, 261]. While some tools exist to support the creation of these articles [254], including *Idyll* (Chapter 4), they typically require that authors are familiar with technical tools like the command-line and often mandate the use of general purpose programming languages like JavaScript or Python. These requirements make it difficult or impossible for those with limited programming knowledge—for example some educators and journalists—to create interactive articles (see Chapter 4). For those who do have the requisite programming knowledge, the article creation process is still time consuming and cognitively demanding (see Chapter 3).

We present *Idyll Studio*, a novel structured editing interface for authoring data-driven, interactive documents. With this tool, authors can use a graphical, WYSIWYG-style interface to create, edit, and publish interactive articles. We show through a first-use study how, in many cases, the tool eliminates the need to use general purpose programming tools, and how it can streamline the article creation process for both novice users and technical experts.

Our contributions can be summarized as follows:

- **Extensions to the *Idyll* framework** to support *reflective* documents which can inspect and modify their own programs at runtime. We use these extensions to create a flexible structured editing API which powers *Idyll Studio*.

- ***Idyll Studio***—a structured editor for authoring interactive & data-driven articles—released as open source software that can be downloaded and used today.[1]

- A **first-use study** in which 18 participants with a wide range of backgrounds used *Idyll Studio* to perform rapid prototyping tasks. We found that *Idyll Studio* enabled non-technical users to complete tasks they otherwise could not, and allowed expert

[1] https://github.com/idyll-lang/idyll-studio/releases

users to create articles more rapidly and with fewer cognitive demands compared to existing tools.

5.1 Motivation & Related Work

Idyll Studio builds on research from narrative visualization [262], computational notebooks [250], literate programming [174], user-interface toolkits [208, 286], and word processors [170]. *Idyll Studio* is a structured editor, and can be seen as a hybrid word processor and interface builder tool for interactive articles.

5.1.1 Narrative InfoVis & Interactive Articles

The information visualization research community has examined the role of narrative in explanatory visualizations, including storytelling approaches [262, 273, 118], the effects of sequence [143] and framing [142], as well as in-depth analysis of usage of different narrative visualization formats such as data comics [32, 34], video [24, 58], and interactive articles [198, 314]. We aim to support narrative visualization authoring in the interactive article format.

A number of systems have been developed with similar goals [200, 254, 293, 179, 218], but existing systems are either limited in their expressivity or require that authors use general purpose programming tools. For example, some WYSYIWG-style editors allow users to insert arbitrary interactive components into web-based articles [66]; however, in previous such systems, components are treated as independent black boxes and cannot communicate amongst one another—a requirement for common design patterns like linked text and graphics [314]. On the other hand, code-based approaches, while sufficiently expressive, have a high threshold for productive use and are often inappropriate for non-technical users like some journalists and educators [179].

Our work extends *Idyll* (Chapter 4), a framework for authoring interactive articles. While *Idyll* supports the creation of sophisticated interactive reading interfaces [86, 85], it still requires that users learn a markup language and use tools like git and the command-line, even to make simple changes to the text of an article. *Idyll Studio* provides a GUI that allows users to perform basic tasks like editing, composing, and publishing documents without requiring general purpose programming tools. *Idyll Studio* reifies [44] the four

parts of *Idyll*'s reactive document model (components, text, state, and styles) in elements of a graphical interface that allows users to rapidly sequence text and interactive elements while allowing them to script domain-specific graphics using visualization and user interface libraries when needed. In *Idyll*, document structure is intentionally separated from the implementation of parameterized components [317]: users write text and arrange components, and then choreograph component interactions by connecting reactive variables to meaningful domain-specific parameters that components expose. Components are created using domain-specific languages like D3 [55] or Processing [242], visualization grammars such as Vega [258] or Vega-Lite [257], or graphical tools that support the creation of interactive graphics and data visualizations. For example, Lyra [255, 316] lets users specify interactive visualizations by example, while Kitty [158] and Apparatus [260] support the authoring of interactive diagrams through sketching and direct manipulation, respectively. Such tools are complementary to *Idyll Studio*, and can be used to create interactive graphics which are then imported into the system and parameterized using reactive variables [82].

Lee et al. [181] articulate a model of the visual data storytelling process. They describe how multiple collaborators (e.g., data analyst, scripter, and editor) work together to create and present data-driven stories in professional settings, which serves as a guideline for the different roles and tasks that a tool like *Idyll Studio* should support. For example, editors can easily edit text and component properties without needing to understand how the components are implemented, and scripters can use the interface to rapidly iterate on a component's implementation and see how changes look in context. We designed *Idyll Studio* with the assumption that the majority of exploratory data analysis would be done prior to the creation of an interactive article and our structured interviews with user study participants (§5.3.3) support this.

5.1.2 Structured Editing

Our system follows a structured editing approach [277], in which authors use an interface to effect changes in the abstract syntax tree (AST) of an underlying *Idyll* program. In contrast to many structured editors (e.g. [137, 176]), our system allows users to interact

with a live running, visual version of their program rather than a structured version of the textual representation. We are not the first to apply structured editing to multimedia documents; for example, the Grif system [237] took a structured approach to formatting complex documents. However, to our knowledge we are the first to support the specification of data-driven, highly interactive hypertext articles through a structured editing interface.

The structured approach eliminates a class of syntax errors [175] (since every change to the AST will result in a valid program) that may make it more beginner friendly, but possible drawbacks include restricting the expressiveness or fluidity of use for expert programmers who prefer to work in unstructured text editors [205]. Sketch-n-Sketch [136] combines unstructured programming with direct manipulation, allowing changes made in either modality to affect the other. Since *Idyll Studio* supports arbitrary components, we do not expose a direct manipulation tool for editing graphics, but rather allow authors to modify any component through general purpose editing instruments [43]. A domain-specific graphics editor like Sketch-n-Sketch could be embedded inside of *Idyll Studio* to allow direct manipulation of a specific set of components using the instrument API we developed (§5.2.2) or used in an independent but interoperable manner (as in §5.2.3).

5.1.3 Runtime Modification of Code & Interfaces

Programs that can inspect and modify their internal structure at runtime support *reflection* [270, 191]. This functionality, available in many modern programming languages [112, 161, 26], allows programs to dynamically adapt based on input from users, environmental sensors, or other sources [123, 199]. To build a structured editor where authors can compose articles with low levels of spatial indirection, we designed an AST transformation to embed *instrumental components* directly in interactive documents and extend *Idyll* to support reflection, allowing these instruments to modify the underlying program as it runs. Put another way, we augment a running *Idyll* program to convert it into its own live WYSIWYG editor.

Hypercard [29] also supported customization of application interfaces from within the application itself, allowing end-users to create multimedia presentations including hyperlinks

and input forms. *Idyll Studio* supports similar features coupled with a reactive variable system, and allows for the rapid creation of layouts linking text and interactive graphics, as well as full control over the look-and-feel of the generated documents. Both systems achieve this through the modification of source code, but researchers have also developed techniques to let end users customize GUIs that do not explicitly support runtime augmentation. For example, pixel-based reverse engineering [100], runtime toolkit overloading [106], and interface attachments [221] enable modification of user interfaces to support augmented interactions [101], personalization [115] and improved accessibility [110].

5.1.4 Notebooks & Literate Programming

The ability to mix code and prose exists in other systems such as computational notebooks [226] and literate programming environments [174]. While some notebook environments support similar features to *Idyll* (e.g., ObservableHQ [215] has a similar reactive runtime and support for embedded graphics), these environments typically target exploratory analysis use cases in which an analyst uses the notebook as a feature-rich REPL to interactively construct visualizations and data transformations to support their analytic needs [227] in a shareable and collaborative format [300]. While Observable and other literate programming environments are centered on programming, our interface centers on direct manipulation, in which authors can compose interactive articles primarily through the use of familiar interactions like point-and-click and drag-and-drop. *Idyll Studio* is designed to support the creation of rich interactive *reading interfaces* targeted at a more general audience than analysts and data scientists, and affords the ability to customize nearly every aspect of the look-and-feel of the final output.

Literate programming environments like Codestrates [239] and Leisure [65] offer a similar editing experience to computational notebooks but offer additional flexibility—for example the ability to customize the user interface—through the ability to use the programming environment to modify its own user interface. By making the entire medium dynamic, literate programs can "blur the line" between authoring environment and application [173]; this technique has been extended to data visualization for ubiquitous analytics [35]. In this

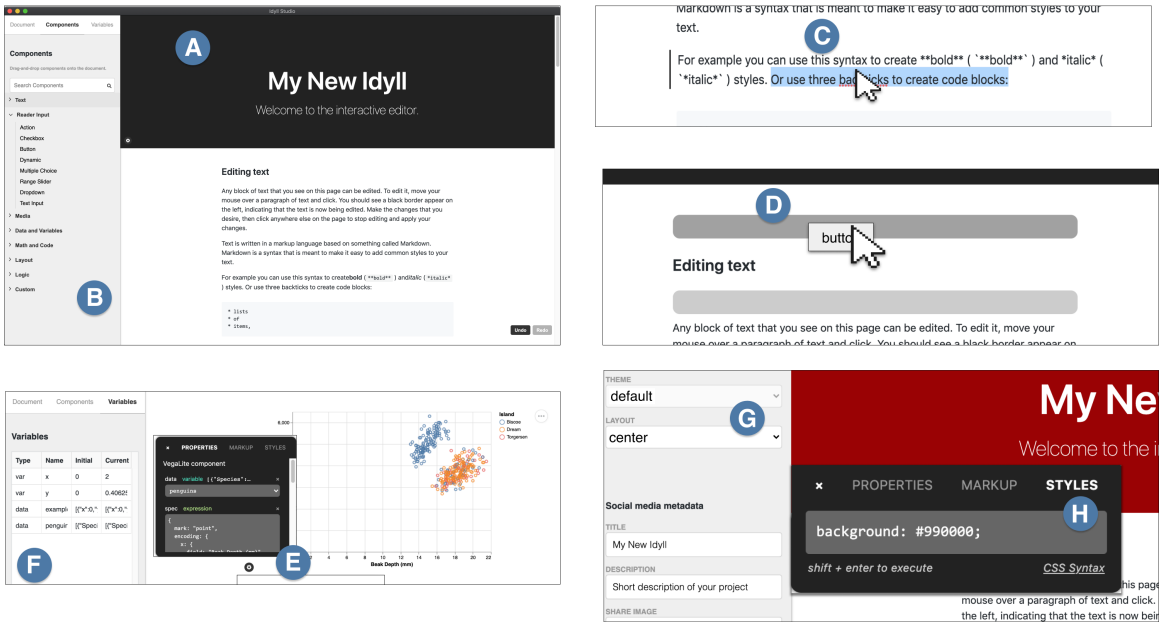


Figure 5.2: *Idyll Studio* utilizes direct and instrumental manipulation to let authors compose and choreograph interactive and data-driven articles. The interface consists of a live article view (A) and an editing panel (B) on the left-hand side. Authors can use text editors (C), insertion points (D), and component editors (E) to construct their articles. The variables panel (F) lets the author inspect and modify the document's reactive state, and document (G) and component (H) styles can be applied to customize the aesthetics.

work, we extended *Idyll* to similarly support reflective documents; however, *Idyll Studio* clearly differentiates between the views for authoring and reading an interactive article. When an interactive article created with *Idyll Studio* is compiled for publication, the instrumental elements used by the authors and designers of the document are hidden from the readers' view by default. Because readers are typically interested in the *content* being presented rather than the underlying implementation of the graphics, model, or user interface (see usage data in Chapter 7), centering these "behind the scenes" elements in a published interface can be a distraction and lead to unnecessary cognitive load [276] for most readers.

5.2 *Idyll Studio*

Idyll Studio is a structured editor for authoring and publishing interactive and data-driven articles. The software is an open source desktop application that supports Windows, MacOS, and Linux. The tool is built on top of the *Idyll* markup language and can be used to edit and view existing articles or create new ones. Programs created with *Idyll Studio* are serialized back to idiomatic markup, compatible with any tools used with *Idyll* markup today.

The interface (Figure 5.2) consists of two major parts: an article viewer A and an editing panel B on the left-hand side of the interface. The article viewer shows a nearly-WYSIWYG view of the article.[2] The editing panel has three tabs, *document*, *components*, and *variables*. Together these elements reify the *Idyll* reactive document model into concrete interface objects; in particular it gives authors access to *text*, *components*, *state* (variables and datasets), and *styles* in order to construct customized interactive articles.

5.2.1 Core Concepts

Compose text and components

A primary task when authoring interactive articles is the composition of text and components. As shown in Figure 5.1, the modified version of the AST that is rendered into the *Idyll Studio* article viewer includes three types of instrumental nodes: *insertion points*, *text editors*, and *component editors*. These three instrumental elements are used to manipulate existing text and components, as well as to insert new text elements or components. To invoke a text editor C an author can simply click on an existing block of text. This will cause a visual indicator to appear, signifying that the selected text can now be edited, and the text displayed will change from rendered HTML to the underlying *Idyll* markup. An author can then directly edit this text (including adding markdown tags or components), and when they click away from the text block, it will transition from the editing mode back

[2]We state "nearly" WYSIWYG because *Idyll Studio* includes *instrumental* components in the interface which allow an author to edit text and component properties and styles. These components do not affect the layout or styles of the page.

to displaying rendered HTML. To use the insertion points D to add new content to an article, an author navigates to the component tab on the left-hand sidebar, and then drags a component from the component library displayed in the sidebar onto the article view. As they do this, gray drop targets will appear in the interface where the component can be placed.

Parameterize components

Next to each component in the article view is a small gear icon which can be clicked to invoke the component editor instrument E. This instrument is used to parameterize components by editing the type and value of existing properties or adding additional properties to the component invocation. When the component editor is invoked, an outline appears around the referent component in the article viewer and an editing window appears below it. The editor's properties tab provides a structured interface for editing component properties, as well as a link to component documentation and the ability to delete the component from the article. Properties can have one of several types, including *literal* (e.g., a string, number, or boolean), *expression* (a reactive expression that references one or more variables or datasets), or *variable* types (a two-way binding between the component property and a variable). By default property values are edited using a text input widget, but this varies by property type: for example, properties with the *variable* type are set via a dropdown containing a list of defined variables that can be bound.

Define variables and datasets

In order to use *expression* and *variable* type properties, authors need to define variables and datasets using the variables panel F. The panel shows a list of all variables and datasets currently defined, their initial and current values in a spreadsheet-like interface, and a button to add additional variables. The cells of this view can be edited directly to change the variable's type (*var*, *derived*, *data*), name, initial value, and current value. The panel also includes a *data* view, which can be used to import datasets by dragging and dropping a file from the author's file system (CSV and JSON files are supported) or

selecting from a list of built-in examples. Once a dataset is added via the dataset view, a new row with the type *data* will appear in the variable view's grid interface. Any variables or datasets that have been loaded in this view can be referenced in *expressions*, and any with type *var* can be used in two-way bindings. A variable's current and initial value start with the same value, but as an author interacts with variable-bound widgets in the article, the current value will update in real time, giving an overview of the article's most recent state. By setting a variable's current value directly, an author can quickly see how components would look under various configurations.

Customize aesthetics

In addition to specifying the composition and behavior of text and components, authors also need to customize the aesthetics of their article [124]. This can be done in *Idyll Studio* through themes and layouts which affect the entire article, or targeted style rules which only apply to specific component invocations. To modify the theme or layout, users can navigate to the *Document* tab in the left sidebar and choose from existing styles or provide their own G. To add component-specific styles, authors can use the *Style* tab in the component editor instrument and add CSS rules to modify, for example, the color, font, position, and size of elements H. In some cases more complex layouts need to be specified via the composition of built-in components. To achieve this, *layout* components can be dragged and dropped onto the article using insertion points, and then manipulated like other components. For example, an *Aside* component can be added to display content (text or another component) in an article's margin.

Access & modify the underlying Idyll markup

In certain cases it will be desirable to access & modify the underlying *Idyll* markup. While most programs can be specified by using the structured editor, certain tasks such as invoking logical components are more ergonomic when done using unstructured text. Take for example the *Switch* component, which conditionally displays its children based on a variable property: to modify the contents of children that do not visually appear in the article,

an author may choose to edit the underlying *Idyll* markup directly rather than manually changing the *Switch*'s value each time they want to edit a different child. Users can select the *markup* button on the component instrument panel to reveal that component's *Idyll* specification. By surfacing only the relevant code snippet directly in the interface, users can make targeted edits without being overwhelmed by seeing the entire program at once. Power users might prefer to edit completely unstructured text as opposed to using our GUI. These users can still access and manipulate the full text of the underlying *Idyll* program via their file system and their programming tool of choice. We envision that supporting these various interfaces will be important to enable collaborations between less technically inclined users, including journalists and educators, and programmers more comfortable with unstructured editors.

5.2.2 Implementation

When users edit text or components through *Idyll Studio*, they are using the interface to make changes to the AST of a live *Idyll* program.[3] In order to support structured editing with minimal temporal and spatial indirection, instruments are placed directly into the document. To do this, we extend the *Idyll* framework in two major ways. First, we developed a transformation (Figure 5.1) that expands the AST to wrap text and component nodes with instrumental elements—such as text and component editors—and add insertion targets, which an author can select through a drag-and-drop interaction. Second, we added support for *reflection*, allowing components to effect changes in the document's underlying program while running. Together, these two features allow us to support structured editing with direct interactions by seamlessly inserting instruments *in situ* in a running interactive article.

Instrumentation

We implemented three instrumental widgets: a *component editor*, a *text editor*, and an *insertion target*. By inserting them into the AST, the instruments appear directly in the

[3]See `https://github.com/idyll-lang/idyll/tree/master/packages/idyll-ast` .

article where the content that they modify appears. While we use the same component editor widget for all components, additional instruments could be added and associated with specific subsets of components. The tree expansion algorithm performs a single breadth-first traversal of the AST: *(1) For each component and text node, replace that component with an instrumental node that has the original component as its only child; (2) for each component and group node, interleave that node's existing children with "insertion point" instruments (components without any children may or may not have an insertion point added as a child, depending on a parameter in the component's metadata).* While this step allows us to render these *instruments* into the document, we need to add reflection in order for them to apply changes to the program.

Reflection

Reflection is the ability for a program to inspect and change its own behavior at runtime. We extended the *Idyll* runtime to support this, allowing the active AST to be changed while the document retains its state. We added an API with a single function `updateAST(newAst)`, that when called will update the actively running *Idyll* program to update to conform to the structure in the new AST. Components can use this function to make changes that modify the behavior of the program at runtime. For simplicity, when passing references to the AST we always use the original, non-transformed copy and make changes to that version before re-running the tree expansion. If these methods are accessed from within an instrumental node, the `nodeId` will refer to the instrument's child, making implementation of the component and text editors straightforward. A suite of helper functions to query and transform the AST is available.

5.2.3 Interoperability

While *Idyll Studio* focuses on specifying the composition and behavior of interactive articles, it may be desirable to utilize other types of editing environments in certain cases, such as when editing a component's implementation (which may involve writing low-level graphics code) or when creating custom vector graphics. While there is an argument to be made that

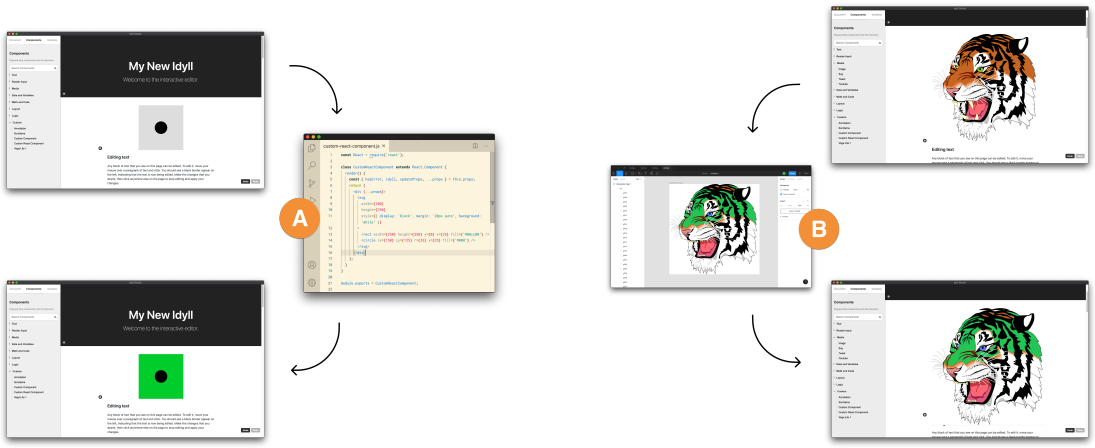


Figure 5.3: Two examples of how *Idyll Studio* interoperates with existing tools: (A) users can click "edit" to call up their system text editor and modify a component's source code; any changes made are immediately reflected in the article view; (B) similarly, users can edit media assets like images or SVGs by invoking a domain-specific editor like Figma and see changes immediately reflected in context.

users might prefer this functionality to exist directly in *Idyll Studio*, we think that there will always be cases where a domain-specific editor provides a better experience than a general purpose one that we could provide. Rather than try to build a monolithic system that supports every type of editing functionality, we instead choose to support interoperability with existing tools that have been highly refined for their specific tasks. Figure 5.3 shows how *Idyll Studio* interoperates with general purpose code-editing and vector graphics editing programs; this functionality can be extended to support other editing environments as well.

In many cases authors want to include their own domain-specific graphics in interactive articles. The most common way of implementing these custom components is through unstructured editing of JavaScript source code, typically in conjunction with a user interface library like React or D3. To support this workflow in *Idyll Studio*, we provide two commands that can be applied to components: *edit* and *duplicate*. To edit a component, the author simply needs to click the *edit* button corresponding to the component that they want to

edit; this will cause the author's system-defined text editor (A) to open with the relevant source file loaded, as well as launch a daemon that listens for changes to the source file and reloads the component inside of *Idyll Studio* as changes are made. This allows users direct access to edit their component's source code without needing to worry about details such as launching a local development server or running a JavaScript bundler to prepare the code for use in a web browser. Each time the source file is saved, the component will instantaneously update in the article viewer, allowing authors to rapidly iterate code changes. If an author wishes to edit a component provided by the *Idyll* standard library, they must first *duplicate* it, and then can freely edit a copy while retaining access to the original implementation.

This interoperability approach is not limited to the modification of component source code, but can be used to streamline the inclusion of other rich media as well (such as vector or raster images) and can be extended to work with any asset that *Idyll Studio* loads from the file system. Figure 5.3 shows how an author can add an SVG component to their article, and then similarly call out to a system-defined SVG editor (B): first, a user clicks an *edit* button that appears underneath the SVG, calling up a system editor based on file-type, and then *Idyll Studio* instantiates a file-system watcher and reloads the file whenever it changes on disk. As the author manipulates the SVG in their preferred editing interface, the view of the SVG in *Idyll Studio* updates automatically to show how the changes look in context.

5.3 Evaluation: First-Use Study

We conducted a first-use study with 18 participants with widely varying technical expertise, ranging from writers with little or no programming experience to professional programmers. Participants completed a short survey describing their familiarity with various technologies, used *Idyll Studio* to complete a series of tasks, and then participated in a semi-structured exit interview.

5.3.1 Methods

To find participants for the study, we advertised it on a social media account associated with the *Idyll* project, a support chatroom for *Idyll* users, as well as a public Slack channel

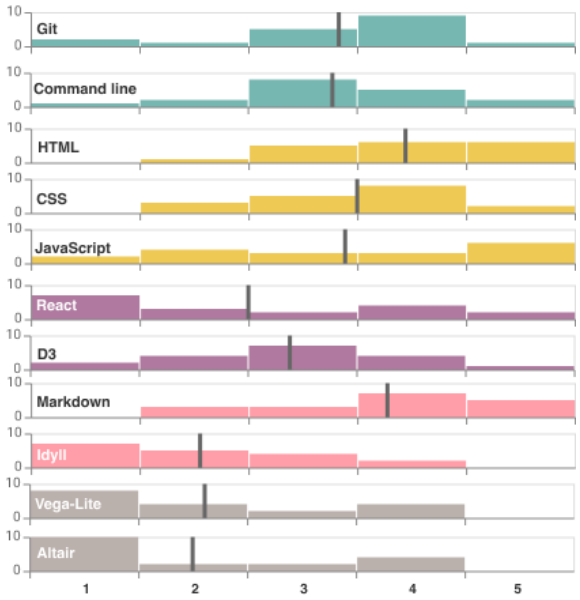


Figure 5.4: The distribution of self-reported technical expertise of our user study participants by category: developer tools, web technologies, JavaScript libraries, markup languages, and visualization grammars. Overlaid vertical lines convey the mean value.

for data journalists. We also invited respondents to share the sign-up link with interested colleagues. Each participant completed a short survey in which they rated their familiarity with various technologies on a 1-5 scale, and listed their responsibilities at work and on personal projects.

We ultimately conducted user studies with 18 people. This cohort worked on a variety of relevant tasks (writing, editing, designing, programming), lived across several continents (North America, Europe, and Asia) and used all three major operating systems (Windows, macOS, and Linux). We asked respondents to report their expertise on 11 technologies commonly used in web development, data science, and the construction of data-driven articles on a scale from 1 to 5. Responses are shown in Figure 5.4. Our study participants have a broad range of familiarity with these technologies, from almost no familiarity to expert levels.

All study sessions were performed remotely over video chat. Each participant was given

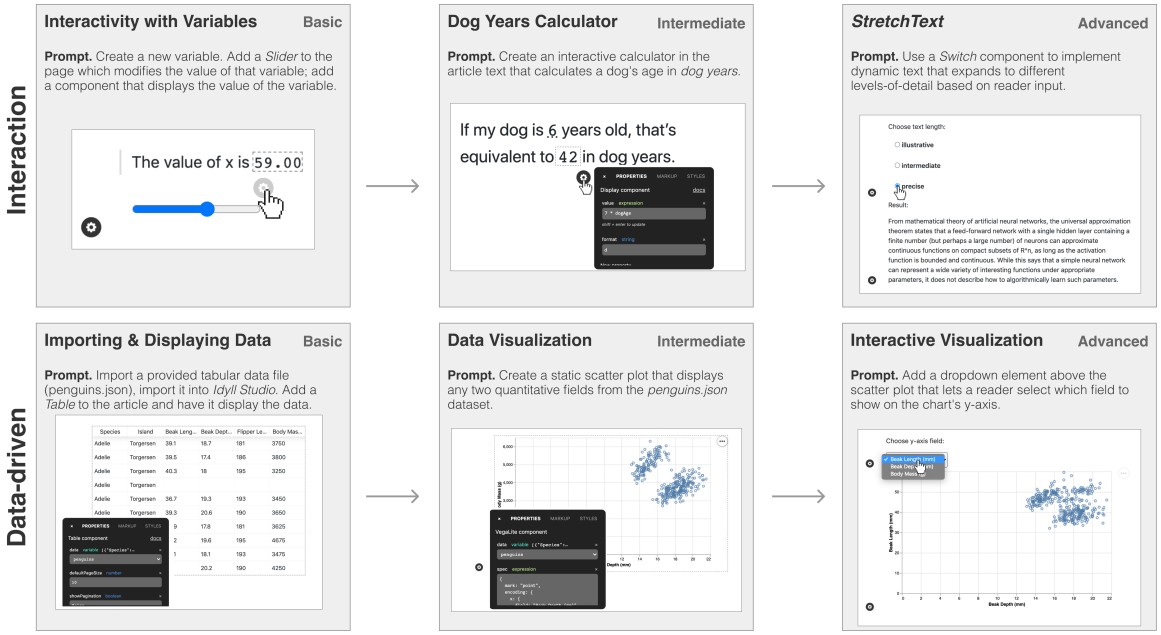


Figure 5.5: Our first-use study asked participants to complete a series of tasks that involved adding interactivity and data-driven elements to their articles. The tasks were chosen to represent common design patterns found in interactive articles.

a link that they could use to download a development version of the *Idyll Studio* desktop application. Participants were given a 10 minute tutorial on how to use the basic functionality, and then were given 30 minutes to complete a series of tasks. The tasks were divided into two categories (*interaction* and *data-driven*) with a basic, intermediate, and advanced task within each category. Participants were given the tasks in order of difficulty, starting with the basic interactive task and ending with the advanced data-driven task (if time allowed) and asked to "think aloud" [111] while they worked through them. After completing the tasks, we performed a short semi-structured interview with each candidate, discussing their experience using *Idyll Studio*, their experience completing similar projects in the past, and if they would consider using *Idyll Studio* in the future.

Interaction Tasks

The three interaction tasks represented common patterns that occur in interactive articles, starting off with a simple example where the participant needed to create a variable and then bind the value of the variable to a *Range Slider* (two-way binding) as well as to a *Display* component (one-way binding), so that when the range slider was manipulated, the display component would immediately show the new value of the variable. The next task was inspired by Victor's *Explorable Explanations* [291] and required that participants build a "dog years calculator," in which text would dynamically update to show the age of a dog in both human-years and dog-years in response to user input. Finally, participants were required to implement *StretchText* [211], where a paragraph of text expands or contracts to display text containing different levels of detail in response to reader input. The *StretchText* task was chosen as it requires that users drop down into unstructured editing of *Idyll* markup.

Data-Driven Tasks

To complete the data-driven tasks, participants were sent a link to a tabular data file containing information about penguins in several quantitative and categorical columns. The participants were first asked to load this data into the system, and then add a *Table* component to the page that displayed the rows of this dataset. After this, they were asked to construct a simple scatter plot using Vega-Lite [257], showing their choice of quantitative fields on the x and y axes. Participants were then asked to modify this scatter plot to make it interactive, allowing readers to choose which field appeared on the y-axis of the chart. To do this, participants needed to create a new reactive variable, bind it to an input widget, such as a radio button selection, and update the Vega-Lite specification to reference this reactive variable.

5.3.2 Quantitative Results

The results of the user study are shown in Table 5.1. All of the participants were able to complete both of the basic tasks and at least one of the intermediate tasks: only one

participant didn't complete both intermediate tasks. The completion rate of the advanced tasks was lower, with 13 out of the 18 participants completing both of those tasks. The "Tech." column in Table 5.1 shows the average value of the participant's self reported score across the 11 different technologies that were asked about in our pre-study survey, which serves as a proxy for a participant's overall comfort with programming and technical tools in general. The table is sorted from highest to lowest *tech* score and shows that a participant's prior familiarity with web development or data science tools was highly predictive of their performance during this user study.

This may indicate that the interface is still not intuitive enough for non-technical users, or that these users needed more time to familiarize themselves with the concepts involved in creating interactive articles. The participants had only 10 minutes of tutorial and 30 minutes to complete all of the tasks. It is encouraging that even the less technical participants were able to complete the intermediate tasks because it suggests that they have an understanding of the basic ideas needed to complete these tasks, but may need more time to synthesize how to compose them in the interface. For example, the advanced interactive data visualization task built directly on concepts that were needed to complete the previous tasks (defining variables, connecting variables to input widgets, referencing variables in expressions, and using Vega-Lite specifications). Given that participants were comfortable using these ideas in isolation, these more advanced tasks may be learnable for these users given more time with the system, and are likely in their zone of proximal development [298].

5.3.3 Qualitative Results

Immediately after spending up to 30 minutes completing the tasks described, we conducted semi-structured interviews where we asked participants to discuss their experience with the tool. The interviews ranged in length from 15 minutes to about an hour. We prepared three areas of questions for each participant: *What did you find confusing or counterintuitive using the tool? Describe the last time you worked on a project involving the communication of data. What tools did you use? What was the workflow? Could you see this tool fitting into your personal workflow in the future? Why or why not? What about in a collaborative*

workflow? We engaged in open discussion with participants based on their responses to elicit more feedback, better understand their needs, and ask how the system did (or did not) work for them. We used an open coding procedure to analyze the interviews and synthesized the resulting codes into several major themes.

1. The interface enabled users to rapidly create designs with less stress. Despite some users not completing all the tests, the general sentiment was positive and participants expressed that they valued the ease-of-use of the system regardless of their technical abilities: "What I really like is to drag and drop [components] and then wire them up with point and click" (P15), "It's fully something I would use - I don't have the time or energy to code from scratch" (P17), "Adding the interactivity and putting it into text - if you wanted to do it in another way you could but it's more work and more stressful" (P9), "Really powerful and easy to use. I feel empowered" (P5). The interface was frequently described as intuitive and participants appreciated that the ability to publish to the web after they had completed an article: "You've smoothed so many of the tough things away from what I usually do. Publishing is a pain in the butt and hey — it's just done!" (P3).

2. But some users struggled to synthesize multiple concepts. Others needed time to understand the reactive model. While nearly all participants completed both basic and intermediate tasks, five less-technical users failed to complete any of the advanced tasks in the allotted time. Difficulties typically occurred when users needed to perform unstructured editing of an expression or manipulate a component's property type. All users were comfortable writing simple one-line expressions, even ones that referenced variables (for example, an expression like `dogAge * 7` was needed to complete the dog years calculator). However, when needing to add a reference to a variable in a more complex expression, as in a Vega-Lite specification, some users struggled and felt overwhelmed: "I didn't feel comfortable or grounded in this part of the test. I just felt like I was trying different things and seeing how the graph would respond" (P18). Participants expressed that they needed some time to internalize how the interface worked: "Everything logically made sense but it took some time to think through things" (P17), "I needed some time to clarify" (P8). For some this did not last long ("I came in with a slightly wrong preconception, but that was all of 30 seconds or so" (P3)), but others needed more time ("I would have to do it a few times to

	Role	Tech.	I1	D1	I2	D2	I3	D3
P1	comp. biologist	4.2	y	y	y	y	y	y
P2	programmer	4.2	y	y	y	y	y	y
P3	hci researcher	3.8	y	y	y	y	y	y
P4	vis. practitioner	3.7	y	y	y	y	y	y
P5	hci researcher	3.5	y	y	y	y	y	y
P6	journalist	3.4	y	y	y	y	y	y
P7	hci researcher	3.4	y	y	y	y	y	y
P8	lab manager	3.4	y	y	y	y	y	y
P9	programmer	3.2	y	y	y	y	y	y
P10	vis. practitioner	2.9	y	y	y	y	y	y
P11	editor	2.9	y	y	y	y	y	y
P12	cs/journ. student	2.5	y	y	y	y	y	y
P13	data scientist	2.5	y	y	y	y	n	n
P14	analyst	2.3	y	y	y	y	n	n
P15	designer	2.3	y	y	y	y	n	n
P16	writer	1.8	y	y	y	n	n	n
P17	ux researcher	1.8	y	y	y	y	y	y
P18	designer	1.3	y	y	y	y	n	n

Table 5.1: We conducted a first-use study with 18 participants. The *tech* column represents a participant's average familiarity with various related technologies; the subsequent columns each represent a task: interactive (*I*) or data-oriented (*D*), and basic (*1*), intermediate (*2*), or advanced (*3*).

get that to work" (P15)).

3. The interface should provide more visual feedback, promote experimentation, provide documentation on demand, and have guardrails to prevent likely mistakes. To make the interface more forgiving and promote learning, the interface should do more to prevent mistakes and guide users in the right direction. One designer without programming experience described a desire for more visual feedback: "Since I'm so unskilled, all the visual cues I could get would be appreciated. I was just operating on a really crude understanding of algebra and how things mapped together. I would definitely take all the hand holding I could get in the UI" (P18). The different types that properties could have were not adequately explained or discoverable in the interface: "[The types] could be more intuitive. It's

not obvious what they mean or how to navigate them" (P11). The interface shined when it promoted experimentation ("In *Idyll Studio* everything seemed to work very nicely. You could fiddle around and get things working quite easily" (P14)) but sometimes users would unintentionally overwrite an important part of their document, for instance the dataset that they had loaded (P6, P15).

4. The interface empowered users and provided better support to less technical users compared to existing tools. Despite the issues, participants found value in what the interface allowed them to do compared to existing tools. Some participants could not replicate what they had done in the tasks with tools they currently use: "I would paste a screenshot of a graph into a deck" (P18), "Sometimes I use PowerPoint, but it is limiting. I can write a little HTML too but that is also limiting" (P18). Others found existing tools frustrating: "I don't have a good workflow. I've tried other tools and just gave up. I have used Observable notebooks—it's a bit confusing, I get overwhelmed using their stuff" (P17). Despite the learning curve of *Idyll Studio*, users found the drag-and-drop interface and structured editing easy to work with, and were excited by its potential: "Once I took the time to learn it would be useful" (P18), "It's not a completely entry level tool—at least there are uses that are not entry level. For people who develop some facility it can be very powerful" (P8).

5. Participants valued ownership of code, data, and styles, and the tool fit well with existing workflows. Several of our participants noted that they appreciated that the application was distributed as a desktop application so they could use it with private data files and without relying on third-party infrastructure (P4, P7, P8). The ability to take the generated source files and send them to other tools was also important. Participants—especially journalists and researchers who frequently published their work online—wanted the ability to host their articles wherever they wanted and have complete control over the look-and-feel (P1, P3, P6, P11, P15). The fact that *Idyll Studio* could be used to customize styles and layouts, and produced web-standard HTML, CSS, and JS files that could easily be accessed, were crucial features for these users. As we expected, most users treat the tasks of exploratory analysis and narrative visualization separately, and when creating interactive articles typically start with clean JSON or CSV files to be used in their article (P3, P6, P7, P9, P10, P12, P13). However, there were some requests for more data transformation

utilities; several participants wanted to have access to a data-frame API (P2, P3, P8) to make it easier to access and filter data field names in expressions.

5.4 Evaluation: Expressiveness

While the first-use study is a good indication that the editor can be used to specify small sections of self-contained interactivity, it is also important to understand how the interface scales to more complex articles. Here we describe how *Idyll Studio* can be used to implement common *Scroller* and *Stepper* patterns [198] (Figure 5.6), and to reimplement an existing explorable explanation (Figure 5.7). These examples are available online.[4]

5.4.1 Scroller & Stepper

A common design pattern in narrative visualization is to animate through a series of visualizations, each of which has a snippet of corresponding text. This can be presented as a slideshow requiring the user to click through each step, or triggered via a scroll interaction. To implement the scroll version, an author would first drag the *Scroller* from the components tab of the left-hand panel onto the document. This immediately adds an example to the screen with sections of text that scroll over a chart in the background. The chart specification can be parameterized using the scroll position by creating a new reactive variable and binding it to the Scroller's 'currentStep' or 'currentPosition' property, then using that variable in the chart specification. The text associated with each step is directly editable via the *Text Editor* instrument. To add steps, they can be dragged onto the article from the components panel. A *Stepper* design can be implemented in a similar way: the component exposes a similar API to bind a reactive variable, and existing text can be edited in place, although new steps must be added by unstructured editing of the Stepper component's *Idyll* specification.

---

[4] *Scroller/Stepper*:
`https://idyll.pub/post/scroller-stepper-example-81e37de0fde8487fdb64c378/`
Barnes-Hut:
`https://idyll.pub/post/barnes-hut-re-implement-e0587578480b3cc37f89ec62/`

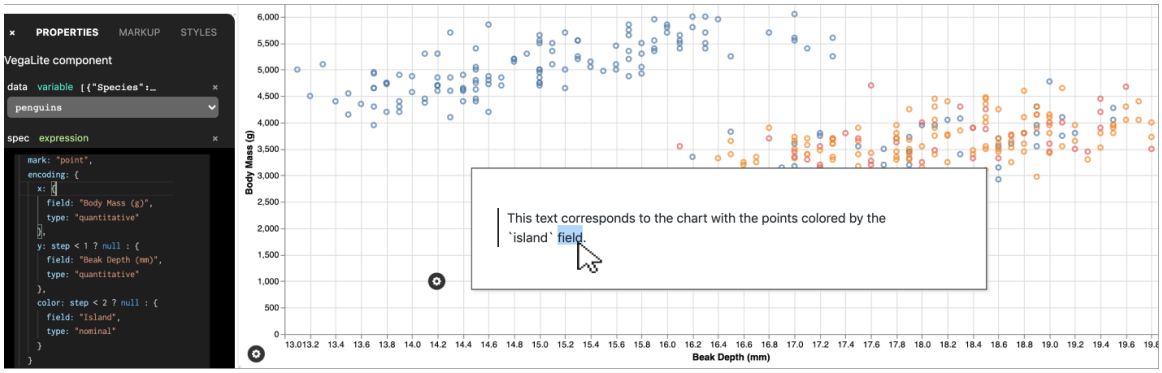


Figure 5.6: Specification of a *Scroller* layout, which displays a sequence of text blocks over a visualization that updates as a reader scrolls. Text is edited in place and the chart's specification is parameterized with a reactive variable that tracks scroll depth.

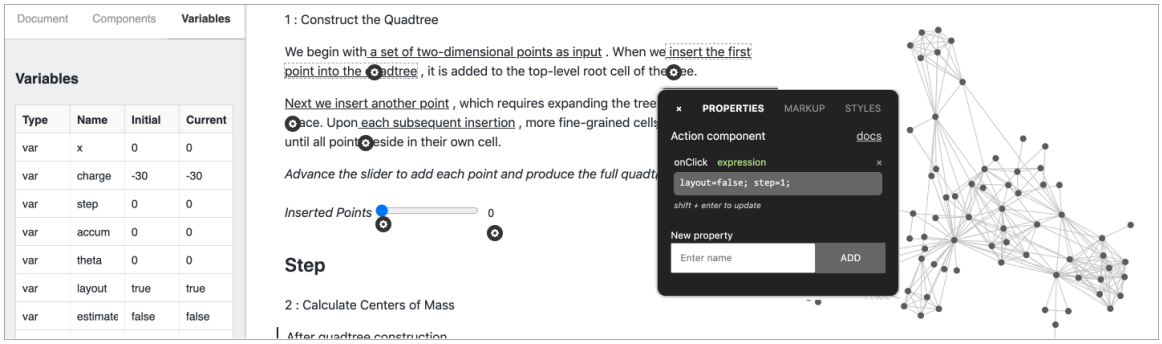


Figure 5.7: A view of *The Barnes-Hut Approximation*, an existing explorable explanation reimplemented using *Idyll Studio*. A graphic stays fixed to the right side of the page as readers scroll, and responds to links and input widgets placed in the text.

5.4.2 The Barnes-Hut Approximation

The Barnes-Hut Approximation is an existing *Idyll* document that presents an explorable explanation of an algorithm used to perform force-directed graph layout calculations. The article consists of a fixed graphic on the right-hand side with text that scrolls by on the left. As the reader progresses, the graphic updates in response, and parameters can be manipulated through controls embedded directly in the text. We downloaded the article's graph visualization component and reimplemented the layout, text, and interactions of the article through the *Idyll Studio* interface using direct manipulation, structured editing, and simple expressions to handle input (typically things like 'layout = false; step = 1;' in a click handler) and create links between components and sections of text. To recreate the layout, we used the drag-and-drop interface to add a *Fixed* component to the right margin and embed the graph visualization inside of it. We then used the *Action*, *Display*, *Range*, and other components to link sections of the text to specific states in the graph, and elicit user input to specify algorithm parameters. The exercise shows that the user interface can scale to support more complex narrative visualizations, though there were some pain points: the component gear icons may need to be positioned in a more structured way when there are many components on the screen to avoid overlap; we also discovered some minor bugs in our markup serialization implementation. It can be difficult to edit a side-by-side design on a small display—in general, we have not optimized *Idyll Studio* for screens smaller than a 13-inch laptop.

5.5 Future Work

We presented *Idyll Studio*, an expressive WYSIWYG-style editor for authoring interactive articles. We conducted a first-use study in which 18 participants were asked to perform rapid prototyping tasks and validated the utility of this type of GUI environment for a broad range of users to author interactive articles. Our system helped both technical experts and non-technical users create interactive articles, making the process less stressful and lowering the barrier to entry by reducing the complexity of the task and limiting the need to use general purpose programming tools. The participants appreciated being able to use familiar *point*,

click, and *drag* interactions to author interactive articles without needing to learn a new language or memorize a library of components. We found that the system fit well with users' existing workflows and supported them better than existing tools, although there was a learning curve for less technical users. We also demonstrated the expressivity of the system by implementing design patterns commonly found in interactive articles, and by using it to recreate an existing explorable explanation.

Idyll Studio is a structured editing environment for authoring interactive and data-driven articles that reifies the four parts of a reactive document model (text, components, state, and styles). In order to create this system, we extended *Idyll* to support reflective documents and developed an AST expansion algorithm that allows us to augment any *Idyll* document with instrumental components, allowing for structured editing of the document using interface elements that reside directly in the document itself. This functionality is accessible via a flexible API which powers *Idyll Studio*. While general AST transformation is a mainstay of programming languages, to our knowledge it has not previously been utilized for reflective WYSIWYG interface design tools. The approach is conceptually simple, powerful, and generalizable, and so could be adopted by future structured design tools.

There is still much work to be done in this area. *Idyll Studio* provides an open-source platform for implementing and testing new techniques, and other researchers may use *Idyll Studio* as a starting point for further extension (e.g., support collaborative authoring, provide guidance to authors, flag likely mistakes, etc.) or integrate it with new research systems or component authoring tools. Despite the positive results of our user study and our findings that the system lowers the threshold for less technically inclined users, the system could still do more to limit the amount of unstructured text editing that needs to be done and make specific concepts of the interactive document model (for example, component property types) more intuitive. Certain interactions can not easily be specified in our tool without writing markup. Identifying patterns that allow users to author such interactions entirely via point-and-click is an interesting area of future research that would continue to benefit interactive article authors.

Chapter 6

FIDYLL: A COMPILER FOR CROSS-FORMAT DATA STORIES & EXPLORABLE EXPLANATIONS

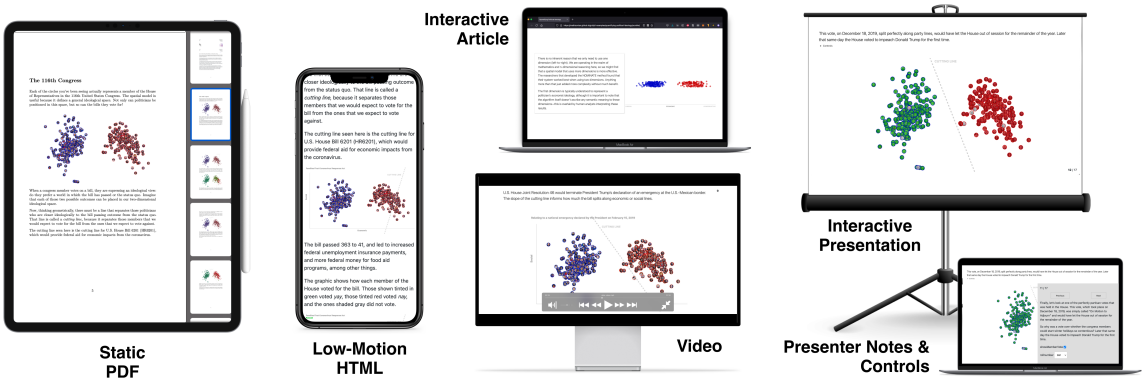


Figure 6.1: *Fidyll* supports serializing five different output formats from a single source document. Here different versions of the case study *Quantifying Political Ideology* are shown. The different formats each have their own strengths and weaknesses.

Interactive articles are a powerful medium for conveying complex, data-driven stories to wide audiences in engaging and understandable ways. The format is used by major news outlets like the New York Times (e.g. [223]), non-profits and other advocacy groups [240], and scientific communicators [102, 219]. These visual narratives can utilize dynamic techniques like personalization [19] and self-reflection [167] to increase audience engagement [89] and improve learning outcomes [194]. However, they tend to be time consuming and difficult to produce (See Chapter 3).

Moreover, there are other formats (or *genres*) of narrative visualization [262]—such as slide shows [198], lectures [248], data videos [24], and comics [33]—each of which may be more appropriate for use with certain audiences or contexts. For example, while a rich interactive narrative like *Snowfall* [60] may be engaging to some audience members [198],

others may find the extensive use of animation to be distracting or even disorienting [113]. A researcher who publishes a new finding as an interactive article may wish to deliver the same results in the form of an interactive presentation (e.g., [85]); a data journalist who publishes an interactive data story online may need to create a static version of that same article for publication in a print newspaper or in animated format to be shared on social media (e.g., [86]).

While it is clear that there is no "one-size-fits-all" format for narrative visualization, authoring tools don't take this into account: existing tools like Ellipsis [254] and *Idyll* (Chapter 4) focus on producing a single artifact—whether it be an interactive article or slideshow, or an annotated visualization—and offer limited support for authors to produce alternative versions. In this work, we explore an authoring tool that facilitates the re-targeting of interactive, data-driven content across multiple formats. With *Fidyll*, authors write their data story in a high-level markup language and the system compiles this into a standard data schema that can then be used to produce a number of different formats including interactive articles, interactive slideshows, videos, static PDFs, and low-motion web pages. Our contributions include:

- We conduct semi-structured interviews with seven domain experts to understand the requirements and workflows associated with multi-format interactive articles. We synthesize findings through an open coding process and use these results to construct motivating usage scenarios.

- We build *Fidyll*, a cross-format compiler for interactive data stories and explorable explanations. *Fidyll* produces interactive articles, slideshows, PDFs, and videos from a single input source. *Fidyll* is open source and available for use at `https://github.com/idyll-lang/fidyll`.

- We produce three articles, each of which corresponds to a motivating scenario. Through these case studies we show that *Fidyll* provides a large amount of expressive leverage by allowing authors to produce multi-format articles while reducing the overall amount of non-narrative markup by up to 90%.

Format	Interaction	Animation	Benefits	References
Scroller	✓	✓	Highly polished designs may elicit more engagement from readers.	Gruessing & Boomgaarden[124]; Conlen et al. [89]
Stepper	✓	✓	Supports both synchronous (live presentation) and asynchronous (reading slides as web page) communication.	McKenna et al. [198]; Zhi et al. [314]
Low-motion	✓	×	Graphics are distributed over space instead of time. May support interactivity. Limits motino sickness.	Frederick et al. [113]
Video	×	✓	Well-supported format with powerful distribution platforms like YouTube.	Amini et al. [24]; Bradbury & Guadagno [58]
Scientific Paper	×	×	Archivable, standardized format	Tversky et al. [283]

Table 6.1: Interactive document formats like scrollers and steppers are useful because they can support both animation and user interaction, but other formats for narrative visualization may be more appropriate depending on the context and audience.

6.1 Background

Our work builds directly on research in narrative visualization and interactive visualization authoring tools. We draw inspiration from past work on adaptive layout and re-targeting of content across different display sizes and modalities. Our work is also related to accessibility, archiving, and personalization in narrative visualization.

6.1.1 Narrative Visualization & Interactive Articles

Segel & Heer [262] identified a set of seven genres of narrative visualization (magazine style, annotated chart, partitioned poster, flow chart, comic strip, slide show, and film/video/animation). The set of genres has since been expanded to include additional techniques like interactive articles which use scroll-based triggers [198]. Segel & Heer note that the genres "vary primarily in terms of (a) the number of frames and [...] (b) the ordering of their visual elements." We leverage this insight in order to support targeting multiple narrative visualization genres as output from a single input source. Victor described *explorable ex-*

planations [291] as a technique to combine text and interactive graphics to promote active reading behaviors [21].

Interactive articles are useful because they support animation and interactivity, allowing authors to take advantage of multimedia learning techniques [194]. However, they are not preferable in all situations or contexts (see Table 6.1). Researchers have found that individual preferences play a key role in audience engagement. For example, McKenna et al. [198] found that some readers prefer slideshows to scroll-based articles and engaged more if articles were presented in the format of their preference. Interactivity and animation also present challenges in accessibility: some readers of websites which utilize scroll-based interactivity and parallax will experience motion sickness [113]; interactive graphics require additional development to support screen readers and may not be beneficial for visually impaired users [163]. Additionally interactive articles present a challenge for preservation and archiving; while preservation tools [160] and formats [204] for rich interactive web content exist, they are seldom incorporated as part of the publishing process [62] and are not as well supported as static document formats like PDF [122].

Tools like Ellipsis [254], *Idyll*, and VizFlow [275] were created to aid authors in the narrative visualization production process. Ellipsis and VizFlow are relatively high-level but target a limited range of outputs; *Idyll* is more expressive but requires more markup to achieve similar results; the relationship between *Fidyll* and *Idyll* is akin to that of Vega-Lite [257] and Vega [258], with *Fidyll* serializing much of the low-level *Idyll* code necessary to implement common design patterns. Like Ellipsis, *Fidyll* utilizes a scene-based data model; however, our model generalizes to support an arbitrary number of parameterized graphics embedded within a larger narrative structure. In contrast to VizFlow, our tool supports output targets across multiple formats, and *interactive* graphics, which can be manipulated by readers beyond scroll input. Because data stories tend to written by collaborators of varying technical proficiency [181], simple markup languages like ArchieML [274] have been adopted by many of the major newsrooms [7]. ArchieML can be embedded in existing collaborative word processors like Google Docs. Our system utilizes a novel markup language based on ArchieML and of similar syntactical complexity.

6.1.2 Adaptive Layout of User Generated Content

Given the prevalence of mobile devices [89], the question of how to adapt content for various display sizes and layouts is of major importance. Hoffswell et al. [139] described techniques for flexible responsive visualization design, allowing authors to produce visualizations that gracefully adapt between mobile, table, and desktop screens. In this work we treat graphics as parameterizable black-boxes, with the assumption that they can gracefully respond to variously sized displays. More generally, the problem of adaptive document layout has been of great interest to researchers in the HCI community [145]. Jacobs et al. [147] developed a method of grid-based document layout building on the fundamental visual grid-system from the Swiss school of design [206]. In this work we also utilize grid-based templates, but don't attempt to automate fine-grained placement of components. Instead we allow authors to modify the placement of text and associated graphics in terms of the grid when necessary. In the future an automated adaptive layout algorithm could be incorporated.

6.2 Formative Interviews

To understand the needs and practices of interactive article authors we conducted semi-structured interviews with seven domain experts, including a data visualization practitioner and educator (P1), a data journalist and academic researcher (P2), an artist and scientific communicator who uses data visualization and physicalization as their medium (P3), a machine learning and human-computer interaction researcher (P4), a data journalist and human-centered artificial intelligence researcher (P5), a librarian specializing in physical and digital maps (P6), and a machine-learning researcher (P7).

6.2.1 Methods

To source the interview participants we directly emailed three people who were known to be domain experts. We also posted a call for participants in a Slack channel dedicated to *explorable explanations* [291], and on Twitter, where the authors are followed by a number of data journalists and visualization researchers. The interviews took place remotely over a video call, and lasted about thirty minutes. Our interview script (available in the sup-

plementary material) served as a general guideline for the interviews, but we engaged in open discussion with the interviewees, guiding the conversations based on their interests and expertise to better understand their needs. After the interviews were completed we used an open coding procedure to analyze the interviews and identify the most salient themes, presented below.

6.2.2 Qualitative Results

Format is determined by intrinsic and extrinsic factors. Respondents discussed a variety of formats that they had used to present narrative data visualizations, including web articles, slide shows, Power Point presentations, Google Docs, PDFs, data physicalizations, magazine articles, print newspaper stories, and interactive maps. The choices for which format to use were driven by a variety of factors, including format affordances ("How do we educate and teach people the capabilities and limitations of machine learning? One way to do that is through interactive articles, through play, education, and other explorable-type interfaces" P4), audience preferences ("I try to do things as simple as possible... especially for journalistic audiences it's not in your best interest to make a very complicated data visualization", P2), author expertise & resources ("We are all in different backgrounds, but we aren't scientists. We don't have a certain workflow to work with." P3), author goals & preferences ("There are some things that research papers don't encourage... I don't know why reviewers don't like empty space, no matter how pedagogical or good the diagram is. These restrictions are good at times, but if you want to write for an audience that wants to learn I feel slideshows or what *Distill* does... those things are nice for reaching a wider audience." P7), and externally imposed constraints ("At [national publisher] I would make charts for the web and then have to put them in the magazine... It sometimes went the other way too where they had a feature magazine article, some illustrator would have a graphic that works in print, and then we'd make an interactive version of it." P5).

Visualizations from exploratory tools are often repurposed for use in a narrative context. While many authors used JavaScript to create custom web-based interactive visualizations ("The tools are your standard web-development tools" P4), it was also com-

mon for authors to create graphics using exploratory analytics tools like R and Python ("We mostly use R and R Studio. But it depends on the story." P3, "We do the interactive version where we don't know what we're looking for and its very exploratory and then we make an explanatory version" P5.). These graphics may be included directly in articles without additional modification or exported in a vector format for additional manipulation and annotation ("I tend to use R for most of the things that I do and will polish in Adobe Photoshop or Illustrator if needed" P2; "The assets, the coupling between them, you have some representation of your interactive thing, you save that out once, and then you work with it in another program: you have your SVG crowbar, pull it out, open it up in Illustrator and hope it mostly works." P5). In some cases this approach makes it difficult to add interactivity post-hoc (P2, P7).

It is common to create static and interactive versions of the same content. Most interviewees had experience adapting materials from an interactive resource to a static one ("For [project] it was visualizing principal component analysis. One of them is in JavaScript on the client, another one is in Python in Deepnote, and then its a PNG in the final report. Three different renderings of the same—it's a huge pain." P1). This was done for a variety of reasons, including academic publishing requirements ("The dissertation format required a PDF at the end of the day." P4), to create an long-living archive of the interactive resource ("Not knowing how long its going to last for, if they're going to have access to the software they used to create it, if they have to present in other formats like a paper, or if they want to go out into the community and work with people who don't have access to digital tools and methodology, how do we show it as a PDF?"), or to create short executive summaries of the main points of their interactive article (P1 "We want the raw data and a data scientist might, but I don't want to hand you that. What I want to hand you is maybe an interactive report you can scroll through, but the mayor doesn't want that. The mayor wants the last thing which is 'here are a few bullet points, who cares what the methodology is, what's the takeaway?"').

Graphics are reused across formats with little modification, but may be omitted from static materials; text may be rewritten to better suit different audiences. When authors needed to adapt graphics for different formats, they typically did so

without making significant changes to the underlying graphic ("There's no special treatment or augmentation [when translating graphics from a static to an interactive version]" P4). When changes were made, it was typically to add annotations ("We'll take screenshots of the graphics, paste them in slides, and write some annotations on top of it" P5), or to capture some interactive aspect of a graphic, for example by recording a video or taking a screenshot ("Authoring interactive graphics, primarily for the web but secondarily for a keynote or slide deck... same annoying problem of how do I capture the example, but at least its not a PDF so you can take screenshots or GIFs or videos to show the interactions more in depth" P4, "To preserve the interactivity you can record a video and add a transcript to show off what the tool once did and provide contextual information about why you made the choices that you did." P6). Interactive graphics may be replaced by small multiples in a static environment ("there were only two states [in the interactive graphic] and you can capture it in a screenshot and show them side-by-side" P4), or they may be omitted altogether due to the lack of support for adapting to the new environment ("we did a web-only version that was a Leaflet map that people could click on; we couldn't figure out a way to make that easily navigable in print" P4). Authors frequently changed the text of their articles to better fit the expected audience of that particular format ("Delivering a narrative summary based on the data science results remains a big problem. How do you go from displaying a bunch of raw data and visualizations to constructing a narrative at different levels of abstraction and expertise." P1).

Authors want to produce more cross-platform content but may not have the capacity to do so. Most of the authors had ambitions to create more cross-format content ("It's very important for our organization to be able to deliver narrative summaries of data science in an automated way... right now there are four assets that we produce." P1). They noted that they understood how different formats are beneficial for different audiences and contexts ("The interaction should facilitate a faster understanding or learning compared to reading a 10 page PDF. How do we give someone an interactive summary to come away with most of the important stuff? The details we can leave somewhere else..." P4). However, participants noted that due to time, resource, and format constraints they often don't do this in practice ("We are not technical people. We are not programmers" P3; "It's work

that is not even seen. If you look at it, I don't think that it matters to you what colors I used or how the elements are arranged." P7; P5 "We're making a nice interactive interface for exploring GAN outputs. It works well.. I can imagine a nice blog post about it, but I'm unsure about how to boil it down to a research paper... it seems so much worse than the blog post version in terms of reading. Why would anyone read this PDF if we could make the version with inline [interactive] examples?").

Existing tools don't promote accessibility or archivability of interactive content. While many interactive article authors were focused mainly on what they could do to improve the effectiveness of their publications work ("It should be very precise... the data representation should be accurate. If there's an interaction it should only help the person engage with the results, come away with a technical insight, or learn something new." P4), others were concerned about the longevity of these materials however they were lacking resources and guidance. ("We'll have a student who spent a semester creating a really snazzy data project... and they're realising that its hard to preserve that type of work as a portfolio piece... This is the type of question that I get. I have been approaching things on a case-by-case basis as there aren't great best-practices established, its really thinking about the goals of the particular researcher and their situation." P6) or how they work for users who may be vision impaired or prefer reduced motion.

6.3 Motivating Scenarios

Based on our interviews with domain experts, as well as the authors' own experience working on data journalism and scientific communication projects, we developed a set of three motivating scenarios grounded in current real-world practices. While not exhaustive, these scenarios are meant to represent the breadth of domains and tasks which demand that authors of interactive narrative data visualizations need to produce content that lives in multiple formats. In §6.5.1 we use *Fidyll* to implement a cross-platform article corresponding to each of the motivating scenarios.

MS1. Data Journalism. A data journalist wants to use a statistical model as an aide in explaining the voting records of members of the United States House of Representatives. She works with a data scientist to identify various parameters of the model that would

make a compelling story. The journalist then writes text that guides readers through these various parameters. The journalist works with a graphics editor to create a visualization that presents the output of the model. The story needs to work on desktop, mobile, and tablet devices, and related static renditions of assets need to be created for a print version of the news paper as well as for use on social media and the paper's homepage.

MS2. Non-profit Advocacy. A watchdog organization regularly publishes reports online discussing the efficacy of various climate change mitigation approaches. These reports are highly technical, consisting of charts, tables, references, and occasionally interactivity. The writers are experts in climate science, policy, and data analysis but they are not experts in web development or graphic design. The organization wishes that all of their stakeholders can participate in the creation of these articles, but currently all of the work falls on to the shoulders of one team member, who happens to have some web development experience. They wish to more easily incorporate the results of statistical models written in R and Python without writing JavaScript and HTML.

MS3. Scientific Publishing. A researcher has identified methods to make a kernel density estimation algorithm significantly faster, but with certain trade-offs in the fidelity of the algorithm's output. The researcher has submitted this work to be published at an academic conference, and needs to write a paper including static images of the output. The researcher will also need to create a slideshow with which to give a presentation, which will be delivered via a pre-recorded video. The slideshow will include animated versions of the graphics used in the paper. In addition, the researcher wishes to publish a digital blog post covering the same topic so that their work can be seen by a wider audience.

6.4 Fidyll

Fidyll is a cross-genre compiler for data stories and explorable explanations. Authors create a single specification which is then used to generate multiple different output formats including interactive articles, dynamic slideshows, data videos, and static documents.

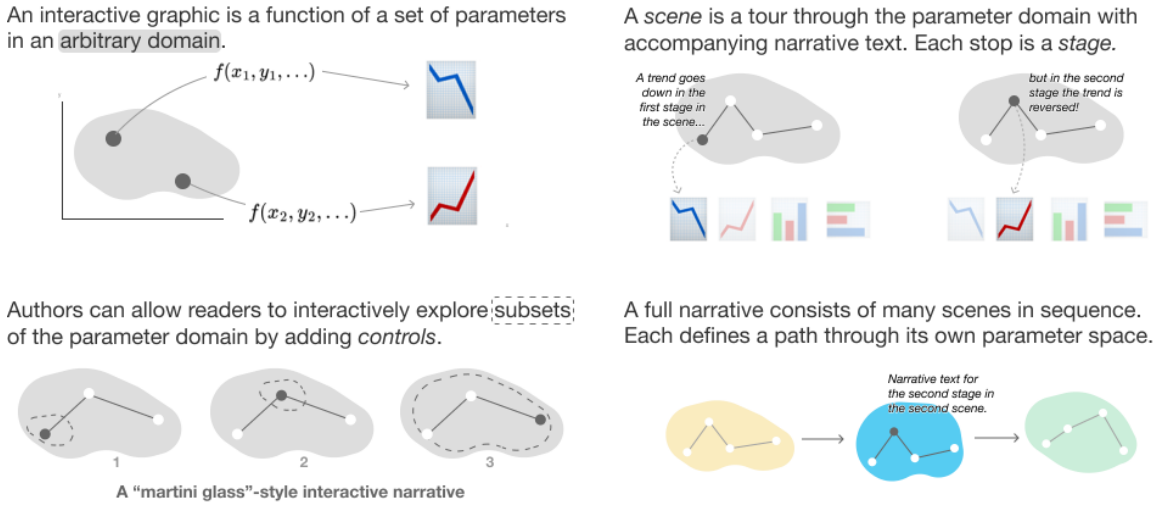


Figure 6.2: *Fidyll*'s data model centers on the notion of giving readers a tour through a high-dimensional parameter space. Each *scene* defines its own parameter space, and each *stage* within the scene defines a set of parameter values. *Controls* give readers leeway to explore parts of this space that aren't directly visited by a stage.

6.4.1 Data Model

A crucial piece of *Fidyll* is the data model (Figure 6.2) used to represent interactive narrative visualizations. By leveraging a expressive and flexible schema, all of the constituent narrative pieces can be specified, independent from their arrangement and layout on the page.

Narrative. The top-level element is the *narrative*. The narrative consists of document metadata such as authors, title, subtitle, etc., along with document-wide information such as pointers to datasets. A narrative consists of a series of *scenes*, which can optionally be book-ended by an introduction and a conclusion.

Scenes. A *scene* is the primary constituent of *Fidyll* data-stories. Each scene consists of a reference to a parameterizable data graphic, along with a list of stages. Scenes may include additional configuration information such as flags to include or exclude particular scenes from particular output genres, allowing authors to customize content on a per-genre

basis.

Stages. A *stage* represents a particular configuration of the data graphic associated with a scene. Each stage specifies a particular configuration of domain-specific parameters for the data graphic, along with corresponding narrative text. Parameters can either be stationary—in which case they take on a specific value as soon as the stage comes into view—or animated, in which case they can loop through a series of values on a timer, or interpolate between two values at the start and end of the stage. Similar to scenes, authors can also use *filters* to include or exclude particular stages from particular output genres. Stages can include a set of *controls*, which allow readers to interact with the data graphic.

Controls. A *control* allows readers to interactively adjust the domain-specific parameters that drive the data graphics embedded in each scene. To add a control, authors specify which parameter it corresponds to, along with the domain of allowed values which readers should be able to specify. *Fidyll* will then infer which type of widget is appropriate and add it to the document.

Animations. Authors can define *animations* for each parameter at each stage by providing a *start* and *end* value, and configuration options (e.g. *duration* and *loopcount*); if the start value is not provided the previous value of the parameter is used. In formats that don't support animation, a series of static frames are generated; authors can define the number and arrangement of static frames with the *frames* (number of frames to generate) and *columns* (the number of columns in the resulting grid) options.

Filters. Authors can provide filters for any scene or stage to specify in which formats the contents will appear. There are four filter keywords that are supported: *include*, *exclude*, *only* and *skip*. The *include* and *exclude* options each expect a list of formats, to specify that a stage or scene will only be included in those formats (*include*) or that it will not be included in those formats (*exclude*). The *only* and *skip* keywords are intended for use during article development, and are boolean flags which allow an author to specify that only one particular scene or stage should be rendered (*only*) or that a particular scene or stage should not be rendered during development (*skip*); we include these based on experience developing long interactive articles, it is often necessary to use such functionality to focus on developing a particular subset of the article.

Output Targets

The primary tasks of *Fidyll* are (1) to parse the author-provided markup into a machine-readable JSON data structure, and then (2) for each of the desired output targets, to transform this normalized data into an *Idyll* abstract syntax tree corresponding to the respective format. *Fidyll* supports five output formats: *interactive article*, *low-motion article*, *PDF*, *interactive slideshow*, and *video*.

Scroller. The interactive article target implements the popular *scrollytelling* layout and reflows to support viewing across display sizes such as on mobile, tablet, and desktop devices. Each *scene* is displayed with the data graphic fixed to the screen while the text associated with each stage scrolls over top. As each section of text appears on the screen, the parameters associated with that scene are applied to the graphic, which updates its display in response. If a *stage* has *controls* associated with it, then the appropriate widgets are displayed beneath the text and readers can manipulate them.

Low-motion HTML. The static article is laid out as a single column of text interspersed with data graphics. For each stage in a scene, the text is displayed with the data graphic rendered with the relevant parameters. Interactive controls are still included, but animations are converted to a series of still frames.

Static Paper. The static article is laid out in one or two columns of text interspersed with data graphics. No widgets for controls are displayed; instead, an appendix is constructed which renders the relevant data graphic with each of the possible configurations of parameters defined by the cross product of controls. To avoid a combinatorial explosion in the where there are many controls, authors can define a subset of configurations which are to be included in this appendix. Animations are displayed as a series of static frames.

Interactive Slideshow. The interactive slideshow implements the popular *stepper* layout and can be used in an asynchronous manner (where the audience reads through the slideshow at their own pace), or a synchronous manner where a presenter uses the slides as support during a live presentation. Each slide corresponds to a *stage* within a *scene*, with the data graphic rendering in a full-screen manner, and the corresponding text (or a summary of the text) displayed above it. A *presenter view* is also generated, which contains the relevant

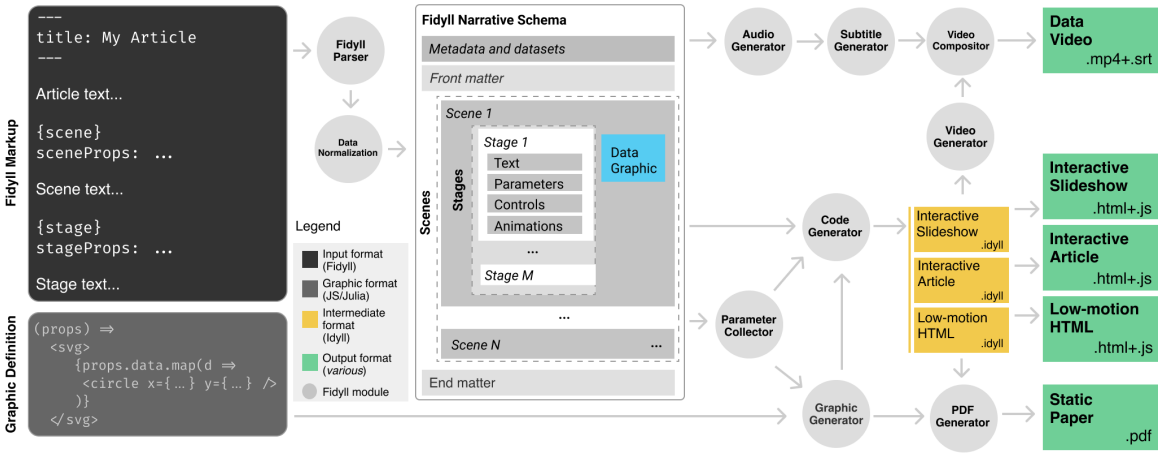


Figure 6.3: *Fidyll*'s software architecture. Authors write narrative text in a markup format and provide parameterized graphics. The markup is parsed and normalized into our schema, which then is used to collect parameter values, generate code, audio, and graphics, and ultimately produce various formats which can be published on the web or elsewhere.

controls for the currently displayed slide; a WebRTC connection is established between the presenter's web browser and the browser of each audience member, allowing viewers to see the data graphics update interactively as the presenter manipulates the controls.

Data Video. The data video target utilizes the same layout as the interactive slideshow. An MPEG-4 video is generated by recording the slideshow played from beginning to end; voiceover is generated by converting the text for each slide to audio via a text-to-speech API and subtitles are generated using the same text in the SRT file format [59]. This output target does not support interactive control but does include all animations.

6.4.2 Syntax & Implementation

Fidyll is implemented as a Node.js package which can be installed and used via the command line on Linux, MacOS, and Windows. The markup that authors write is based on ArchieML, a markup language designed to be human-readable and easy for writers and editors to use. ArchieML is used by both technical and non-technical staff across a number of newsrooms,

including at the New York Times and Washington Post, and so we believe that our variant could be similarly used by authors without a strong programming background. Example *Fidyll* markup is shown below:

```
---
title: My Great Article
authors: Matthew Conlen, Jeffrey Heer
datasets:
  penguins: penguins.csv
---

Some introductory text...

{scene}
graphic: ExampleDataGraphic
parameters:
  booleanVariable: false
  continuousVariable: 0

This text is shown near the ExampleDataGraphic with both variables in
their initial state...

{stage}
parameters:
  booleanVariable: true

The boolean variable is now true, the ExampleDataGraphic has updated to
reflect this.

{stage}
```

```
animations:
  continuousVariable:
    start: 0
    end: 1
    duration: 750
controls:
  continuousVariable:
    range: [0, 5, 0.1]

The boolean variable is true and the continuous variable animates from zero
to one. Readers can manipulate a range slider to modify the continuous
variable after the animation has played, or click a play button to watch
the animation play again.
```

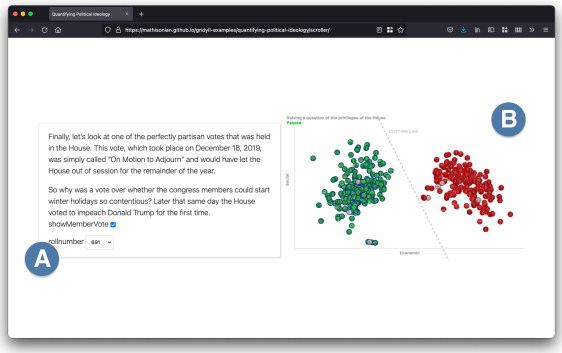
Parsing & Normalization. We built a custom parser to handle the new markup format. It reads the YAML-style markup associated with each stage of the article and builds a JSON data structure to represent the schema shown in Figure 6.3. Because the parameters that authors provide at each stage may be under-specified (authors only need to specify the parameters that *change*), a normalization step occurs after parsing to ensure that the parameter set associated with each stage of each scene fully specifies all relevant parameters.

Parameter Collection. Once the data has been normalized into the expected schema, the parameter collector walks through the scenes and stages to identify all of the possible configurations of parameters that could occur while readers progress through an article. These configurations include not only the parameter values specified at each scene and stage, but also the range of parameter values that could be reached by manipulating interactive controls. This set of possible parameter configurations is used to generate any graphics that are created as images on the server-side, and to create static renditions of interactive graphics to be included as appendices in static versions of the article, allowing us to achieve content parity between the static and interactive output formats.

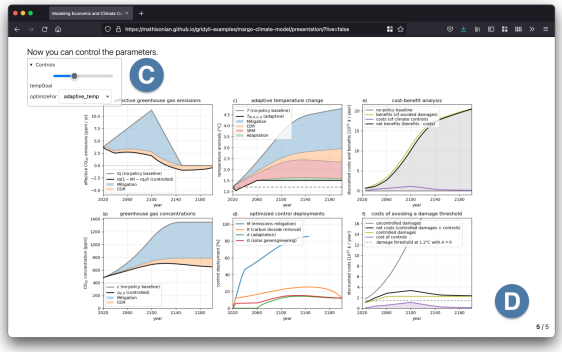
Graphic Generation. While many graphics are generated in the browser using web-based libraries like React and D3, we found that many authors also like to create graphics using server-side scripts written in languages like Python, Julia, or R. The graphic generator calls these scripts with all relevant possible parameter values and generates images which can then be embedded in the interactive article. The image filenames encode the parameter values so that interactivity can be maintained through control manipulation (e.g. as a reader manipulates a slider, the slider's value is used to update the path to the image). The graphic generator is also responsible for making static image versions of interactive graphics that ultimately get included as part of an appendix in a static PDF.

Code Generation. The bulk of the cross-format logic occurs in the code generator, which is responsible for taking in a *Fidyll* narrative and producing *Idyll* markup as output. For each output target, the code generator will iterate through each scene and scaffold the necessary code to generate the relevant layout (e.g. a *Scroller*). *Fidyll* serializes reactive variables corresponding to parameters (each scene defines its own variable scope) and input widgets and animation playback buttons where appropriate. *Fidyll* also defines parameter updates via event handlers; the *onEnterView* event, for example, is used to update parameters when a new stage scrolls into view. Much of the code generation logic can be reused across formats, although each has their own unique way of structuring how the content appears on the page. In the static layout, multiple graphical components are embedded in the article, each with their own fixed set of parameters, rather than a single component being embedded which utilizes updating parameters. The code generator ultimately produces an abstract syntax tree representing an *Idyll* program which is then serialized as markup corresponding to interactive slideshows, scrolling articles, and low-motion articles. This markup is then transformed by the *Idyll* compiler into HTML and JavaScript that runs in web browsers. It is also used as input to the PDF and Video generators in order to produce static formats. Examples of the compiled *Idyll* markup are available as part of the supplementary material.

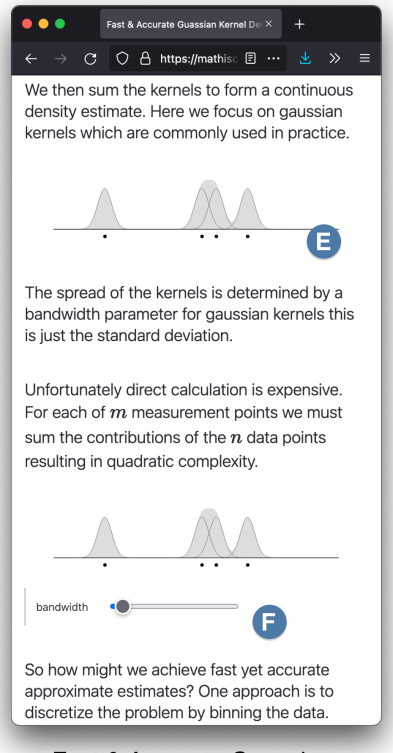
Video Generation. The video generator takes as input the *Fidyll* narrative and runs the text through a text-to-speech API to create a series of audio snippets, one corresponding to each slide in the interactive slideshow. We use the Text-to-Speech service provided by



Quantifying Political Ideology



Climate and Economic Modeling



Fast & Accurate Guassian Kernel Density Estimation

Figure 6.4: We used *Fidyll* to create three interactive articles. In the interactive article version of *Quantifying Political Ideology*, sections of text scroll on the left half of the screen (A) while graphics remain fixed in place on the right (B); graphics update as the reader scrolls and respond to interactions with controls embedded in the text. The presentation version of *Climate and Economic Modeling* provides high-level text summaries of visualizations and embedded controls (C), which can be used to manipulate full-screen graphics (D). In the low-motion HTML version of *Fast & Accurate Gaussian Kernel Density Estimation*, graphics are repeatedly embedded in a single text column with various parameters (E); readers can still manipulate controls to further explore the parameter space (F).

Google Cloud. Once the audio is created, the interactive slideshow is recorded via a headless web browser [8]. The duration of each audio snippet is use to determine the length of each slide in the interactive slideshow as well as the timings for the generated subtitles. The

audio clips and screen recording are composited together to produce an MP4 video and an SRT subtitle file is serialized.

PDF Generation. The PDF generator leverages the low-motion interactive web article as well as the document converter Pandoc [190]. Each graphic in the low-motion article is converted into an image and an HTML file is serialized with the interactive graphics replaced by static images. The HTML file is processed by the Pandoc converter, which produces a PDF using LaTeX.

6.5 Evaluation

We consider this work to be a form of user interface systems research and turn to Olsen's criteria for evaluation [222]. In Section 6.1 we showed the *importance* of the work and in Sections 6.2 & 6.3 we identified specific *situations, tasks and users.* In this section we show the *generality* of our work through a set of three case studies, and show how it *reduces solution viscosity* compared to existing tools by affording authors *expressive leverage*: *Fidyll* significantly reduces the amount of non-narrative markup that authors need to write to generate cross-format interactive articles.

6.5.1 Case Studies

We used *Fidyll* to develop three cross-platform interactive articles (Figure 6.4). These articles were designed so that each would map directly on to one of the motivating scenarios provided in §6.3. The articles cover a range of topics (spatial models of politics; an economic-climate model; an optimization for kernel density estimation) using a variety of commonly used narrative visualization techniques such as a *martini glass* structure and *drill-down* stories [262]. Each of the articles was developed with the intention that it would serve as a useful educational artifact separate from the publication of this paper, which helps ensure that they provide "authentic and realistic motivation" for this evaluation [89]. The articles are available online at `https://idyll-lang.github.io/fidyll-examples/` and a video demo is provided as supplementary material.

Quantifying Political Ideology

In *Quantifying Political Ideology* we provide an interactive explanation of the NOMINATE family of spatial models of rollcall voting for the U.S. legislative branch [232, 231]. DW-NOMINATE scores are frequently cited by journalists who use them as a tool for interpreting political behaviors. The article is written from the perspective of a data journalist who wants to provide their audience with a high-level and intuitive explanation of a topic that is typically presented in a highly technical way. The article consists of three scenes: an *introduction* which explains the basic properties of spatial voting models and provides a high-level motivation for why they are built; *real world examples*, in which the voting behavior and ideological positions of members of the 116th U.S. Congress is explored (readers can interactively select votes which they are interested in learning more about); and a *technical explainer* of the inner-workings of the optimization algorithm used to derive the ideological NOMINATE scores.

MARGO Climate Modeling

MARGO Climate Modeling was designed to match the motivating scenario MS2, where a non-profit organization uses open datasets and scientific models in order to advocate for certain policy actions. In this article we utilize the MARGO model [103] which allows users to specify constraints (e.g. *"keep warming below two degrees celcius"*) and will provide an optimized path for adhering to those constraints through the use of climate controls including mitigation of greenhouse gas emissions, adaptation to climate impacts, removal of carbon dioxide, and geoengineering. The article is a straightforward application of the martini glass narrative structure in which an author-driven narrative is first presented to the reader, walking them through the a series of important points pertaining to the MARGO model, and then allowing them to explore freely within a predefined space of model parameters. Because the model runs in Julia, *Fidyll* determined the multidimensional space of possible parameter values that a reader could reach at any point in the article, and for each possible configurations, ran the model ahead of publication to generate corresponding graphics. In the final version there were 22 possible parameter configurations, which took

Example	Scenes	Stages	Target	Markup	Narrative LOC	Non-narrative LOC	Total LOC	% Narrative	% Reduction (Non-narrative)
Fast KDE	18	54		fidyll	109	320	429	25.41%	
			static	idyll	109	1518	1627	6.70%	78.92%
			slideshow	idyll	109	1083	1192	9.14%	70.45%
			scroller	idyll	109	736	845	12.90%	56.52%
			combined	idyll	109	3337	3446	3.16%	**90.41**%
MARGO	2	14		fidyll	33	95	128	25.78%	
			static	idyll	33	179	212	15.57%	46.93%
			slideshow	idyll	33	219	252	13.10%	56.62%
			scroller	idyll	33	142	175	18.86%	33.10%
			combined	idyll	33	540	573	5.76%	**82.41**%
QPI	2	13		fidyll	57	90	147	38.78%	
			static	idyll	57	495	552	10.33%	81.82%
			slideshow	idyll	57	256	313	18.21%	64.84%
			scroller	idyll	57	190	247	23.08%	47.37%
			combined	idyll	57	941	998	5.71%	**90.44**%

Table 6.2: *Fidyll* reduces solution viscosity by providing authors with expressive leverage. Authors using *Fidyll* write considerably less non-narrative code than with existing markup languages, freeing them to focus on the narrative aspects of their data story.

just under an hour to generate on a 2018 MacBook Air with a 1.6 GHz dual-core processor and 16 GB of ram.

Fast Kernel Density Estimation

The *Fast Kernel Density Estimation* article was developed to represent MS3, the scientific publishing scenario, and was adapted from a presentation given by Jeffrey Heer at the 2021 *IEEE Visualization Conference* [131]. The original presentation was created by having the presenter give the talk into his webcam while screen recording software captured slides produced in Keynote. The slides contained multiple graphics which were developed in JavaScript using D3 [55] and utilizing both Observable [215] and a standard (non-notebook) web development environment. The graphics were interactive, but for the purposes of the presentation multiple video recordings were made with screen recording software and then added to the slides. We were able to adopt these graphics for use within *Fidyll* and repro-

duce the content of presentation as a cross-platform explorable explanation, including an interactive slideshow in which the graphics can be manipulated in realtime by the presenter. To do so we needed to make small modifications to the code of the graphics (approx. 10 lines of code changes in each graphic) so that they adhered to our component API and can adapt to different screen sizes (the graphics are displayed at a different resolutions in different formats, e.g. in the static article they are shown at the width of the text column; in the interactive slideshow they are shown at full screen width).

6.5.2 Expressive Leverage

To assess the extent to which *Fidyll* assists in facilitating the rapid creation of cross-format interactive articles we analyzed the source code of the three case study articles, including both the *Fidyll* markup that authors produced as well as the intermediate *Idyll* files which get created and ultimately compiled into HTML and JavaScript. We compare the *Fidyll* markup to the intermediate *Idyll* markup rather than the final HTML and JavaScript because *Idyll* represents a more realistic baseline in terms of what authors currently use for the task of authoring interactive articles. We computed relevant counts and ratios of the lines of code (LOC) associated with each case study and with the three intermediate *Idyll* formats which ultimately generate the five final artifacts. Table 6.2 shows the results of this analysis. To make the comparison valid across the articles and formats we break each sentence of narrative text onto its own line and ignore any whitespace. The *Idyll* markup that is generated is idiomatic markup that is produced by calling the *idyll-ast* package's *toMarkup(ast)* function. We consider a narrative line of code to be any line in which the majority of the markup is rendered directly on-screen as narrative text; the % reduction metric is defined as one minus the lines of non-narrative *Fidyll* code divided by the lines of non-narrative *Idyll* code.

We found that *Fidyll* reduced the amount of non-narrative code by 82–90% compared to writing the *Idyll* markup for each of the three *Idyll* formats directly. Compared to writing the *Idyll* markup, for a just a single format *Fidyll* still reduced the amount of code written by 33–82% (mean 60%; median 57%). *Fidyll* also improved the ratio of narrative to non-

narrative code that authors had to write: the *Fidyll* markup consisted of 30% narrative code on average, while this number was 15% on average for a single format drafted in *Idyll* and 5% for the combined *Idyll* markup, indicating that the authors would have had to write about 20 lines of non-narrative markup for every sentence in the data story.

6.6 Discussion

We discuss the limitations of our system and study and highlight points of interest to the research community.

Limitations of this work. While we grounded the situations, tasks, and users of our system through formative expert interviews and created a set of realistic case studies, we haven't directly observed our system in the hands of end users. Because our markup language is a dialect of ArchieML, a widely used markup language in newsrooms, we don't anticipate users having an issue learning the syntax. In the future we would like to observe the strengths and weaknesses of *Fidyll* through observation of real world use. While *Fidyll* can express a wide range of common narrative visualization design devices, there are designs that it does not support. For example, the tool expects that readers will progress through content in linear fashion and does not support non-linear *choose-your-own-adventure*-style text. The tool only supports a subset of possible formats and doesn't yet support highly-visual genres of narrative visualization such as data comics or infographics.

The Role of Automation in Data Storytelling. While our system drastically reduces the LOC needed to produce multi-format data stories, there are challenges to the automated approach [130]. For example, the audio narration (generated by a text-to-speech model) frequently mispronounces technical words (e.g. "Gaussian") and in practice we expect that authors will prefer to add voiceover from a human narrator. We attempted to automate other tasks, such as using a language model [63] to summarize text from long form articles to shorter snippets that would be appropriate in formats like a slideshow. We achieved poor results; however, this may be useful to develop further in the future. Our system gives authors access to all of the generated assets so that they can make additional manual changes if necessary.

Parity of Content Across Formats. While there are real educational benefits to

well-designed interactive content (See Chapter 3), there are still situations where static or non-interactive animated content will be preferred. In these cases we believe it is important to still give readers access to the same *information content* [283]. For this reason we ensure that static versions always included additional still images of graphics, representing all of the relevant states that could be reached in the interactive version.

Accessibility of Interactive Articles. *Fidyll* supports authors in producing accessible documents: the static and low-motion formats are preferable for readers who experience motion sickness; the audio and subtitles generated for the video format support readers with low vision. However, there are still major challenges in adapting interactive graphics for use with assistive devices like screen readers [163]; care needs to be taken to generate markup that works well with these devices, and the system should create descriptions of the on-screen graphics for low-vision readers.

System Expressiveness and Design Guidance. While our system is strictly less expressive than *Idyll*, which our high-level markup compiles to, we believe that this trade-off is beneficial for many authors because of the expressive leverage that it provides, and because it provides much more structure in the resulting designs, nudging authors to use common, well-established formats. Many authors of interactive articles are not experts in design and likely benefit from the system making low-level design decisions.

6.6.1 Conclusion

We presented *Fidyll*, a cross-format compiler for data stories and explorable explanations. Our system was informed by interviews with expert users and realistic motivating scenarios. The scenarios were operationalized in three case studies that we analyzed to show the expressive leverage of our system, reducing the amount of non-narrative code that needs to be written by 88% on average.

Chapter 7

CAPTURE & ANALYSIS OF ACTIVE READING BEHAVIORS

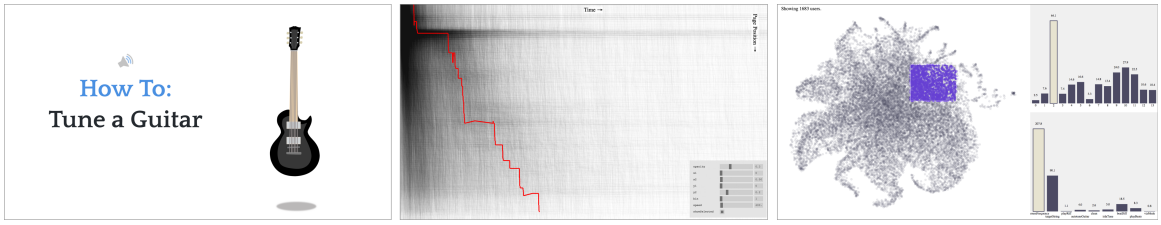


Figure 7.1: An online interactive article *How To: Tune A Guitar* (left) and visualizations of collected reader activity data. *HopScroll* (center) visualizes reader progress over time, revealing reading patterns and fixation points. *Readuction* (right) uses dimensionality reduction of reader feature vectors to enable nuanced segment analysis; linked views show timing and event information for selected points. Along with these tools we present Idyll language extensions for automating the collection of detailed log data, and discuss reading patterns discovered.

The *interactive article* is a form of web content increasing in popularity. Newspapers publish interactive graphics and visualizations in addition to more traditional articles. Educators and technical communicators enrich text with multimedia in an effort to further engage their readers. These interactive pieces often resonate with a wide audience [124] and may more effectively communicate complex topics [194]. Publishers understand that interactive articles can bring both acclaim and a broad readership [138]. However, due to time constraints and a lack of tooling, these articles are seldom analyzed to see if their interactive features are effective at engaging readers and delivering the desired message. Interactive articles suffer from the additional issue that there are no clearly defined metrics or evaluation methods to measure their effectiveness.

Visualization practitioners have debated whether interactive components are effective and worth the significant effort and expense necessary to create them [281, 42, 22]. Researchers have called for more data to be collected on these articles, in a more realistic context than the laboratory [177]. To advance more comprehensive and realistic studies of narrative visualization use, we contribute capture and analysis tools for reader sessions of interactive articles. We first present extensions to the *Idyll* markup language that automate the detailed instrumentation of interactive articles. The logger captures browser metadata, scroll position, and time-series data of mutations to *Idyll*'s article state model, which tightly corresponds to article feature usage.

We next present visual analysis tools to enable exploration of collected reader data. *HopScroll* plots user scroll positions over time to reveal a diversity of reading behaviors. In addition to an aggregate overview, HopScroll shows individual user sessions in an animated fashion akin to *Hypothetical Outcome Plots* [144]. *Readuction* uses dimensionality reduction to visually cluster related feature vectors automatically derived from event logs. Coordinated plots show associated timing and event data for selected points, enabling nuanced segment analysis of reader behavior. Both tools support *micro / macro readings* [282], enabling simultaneous consideration of aggregate patterns and individual behaviors.

We apply our tools in three case studies of interactive articles. We follow the articles through their entire life cycle: from inception through design, publication, and evaluation. Each was written to appeal to a specific audience, and instrumented to record reader interaction logs. These articles were visited more than 50,000 times by readers across the world. We report aggregate usage statistics, compare behavior across mobile and desktop devices, and demonstrate the use of our tools to characterize both article-specific interaction patterns and cross-article reading patterns. Additionally, we show how the collected data can be operationalized and measured against content-specific metrics developed by the articles' authors. We conclude with a discussion of findings, implications, and avenues for future research in this area. By publicly sharing our tools and corpus of reader data, we hope to encourage the collection and study of reader behavior at scale in real-world contexts.

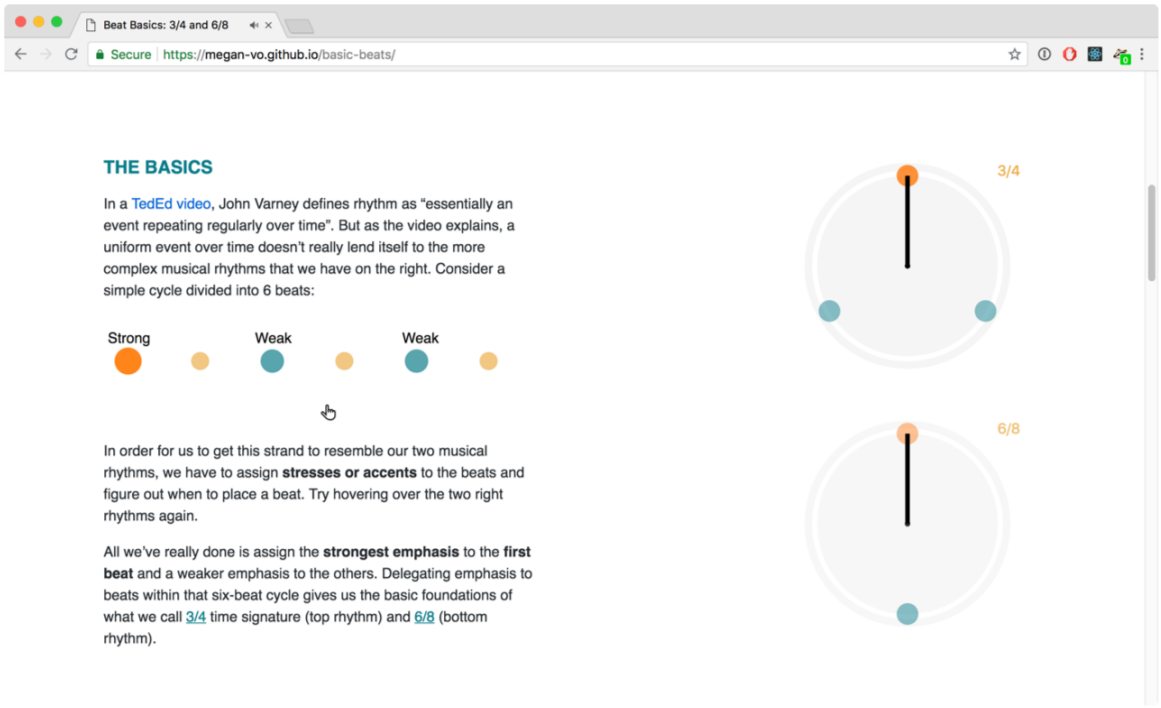


Figure 7.2: *Beat Basics* was produced by Megan Vo, an undergraduate computer science student. She designed it to teach curious readers the basics of rhythmic time signatures in music.

7.1 Motivation & Related Work

Our work is motivated by calls from researchers in the data visualization, journalism, and educational communities for further inquiry into the effectiveness of dynamic content. In conducting our analysis we build on a large body of work on the analysis of webpage usage, and the instrumentation of software applications more generally. We scope our work to consider content specifically that is published online to an anonymous readership with whom the author has no direct communication. We use the phrase *interactive article* to encompass several types of web content. In general these articles are characterized by interleaved text and interactive widgets—often utilizing animations, data visualizations, or simulations—and guide a reader through a primarily linear narrative. This scoping is in contrast to the use of *interactive fiction* by media critics to describe non-linear, branching

text [315], and *interactive non-fiction* which has been used to refer to similar techniques applied in a journalistic context [266]. *Explorable Explanations* [291] are one notable type of interactive article that promote active reading and inquiry into the details of a topic. This type of article is becoming increasingly prevalent on the web [10]. Explorable explanations share design techniques with interactive articles published by news outlets, however they have not been widely studied.

The use of narration alongside graphics has been broadly studied by education researchers. Mayer [194] aggregates principles of multimedia learning that can inform designers of interactive educational content. In a classroom setting the effectiveness of educational content might be assessed via tests or quizzes given directly to students. However, in contrast to a classroom or MOOC setting, we are interested in content published to broad audiences who are not externally motivated, for example, by being enrolled in a class. As such, while the design principles are related, the methods by which materials are evaluated should vary.

7.1.1 Usage Logging and Analysis

There is a rich history of research on capturing and visualizing usage of applications in general [172, 133, 192] and specifically on web sites [132, 229]. Commercial and open source tools like Google Analytics [12], Chartbeat [9], and Matomo [15] are commonly used to collect basic statistics of web page visitation. These services record visitor counts and time spent on pages, in addition to profiling how users flow from page to page. Google Analytics offers web administrators the ability to deploy surveys to website visitors as a means of assessing the quality of page content. In this work we focus on capture and analysis methods that do not require additional page content or input from readers. Other services like Heap [13] focus on tracking behavioral events such as button clicks. Visualization researchers have presented systems for analysis of event logs [97] and clickstreams [186]. For example, CoreFlow [184] is a tool for extracting and visualizing branching patterns in event sequences.

Quantifying user engagement beyond simple measures like page views and time spent

on task is difficult. Attfield et al. [30] propose different facets of user engagement: focused attention, positive affect, aesthetics, endurability, novelty, richness & control, reputation, trust & expectation, and user context. The data that we have collected can be used to assess engagement through aesthetics and richness & control. Rodden et al. [246] contribute a generic framework (HEART) for measuring user experience of web applications. These models are not fully applicable to the study of interactive articles. For example, an article about gun violence [73] may induce negative affect in readers, but still effectively communicate a message. Metrics like retention may not apply to individual articles if there is no need for visitors to return after they have read and learned from the presented content. Some applications opt to tailor metrics and visualizations to their specific domain rather than relying on a general framework. For example, LectureScape [162] visualizes how viewers interact with educational videos and Porta [209] profiles tutoring software. These projects demonstrate that tailoring metrics to a specific domain can lead to more effective analytics platforms. In addition to our domain-specific logging and analysis tools, we work with domain experts to derive quantitative content-specific metrics to assess particular aspects of article design.

7.1.2 Journalism and Narrative Visualization

Visualization researchers have studied the techniques used in interactive and data-driven stories published by news outlets and other publishers. We discussed this research in Chapter 2 and briefly recap it here. Segel & Heer [262] articulated a design space of narrative visualization which was further refined [273] to reflect modern practice. Researchers have noted several opportunities for further research in this space, including evaluating the effectiveness of data-driven storytelling techniques [181]. Along these lines, McKenna et al. [198] report on the effects of design choices on reader engagement in a controlled study. However, Kosara & Mackinlay [177] write that studies will need to be performed outside of a traditional laboratory setting in order to "get stories in front of the types of people who are also the audience for news media." Amini et al. [245] explicate the criteria, methods, and metrics that may be used when evaluating data driven stories. They, too, suggest that

case studies may be necessary in order to better understand how tools and stories perform in the wild. In this research, we conduct evaluation of interactive articles published in an uncontrolled setting. Boy et al. [56] present a case study examining the effect of the presence of narrative text on users' engagement with visualizations in the wild. Boy et al. use semantic operations to operationalize log data by mapping it onto a generalized vocabulary of user actions. While this work is similar to ours in its collection of usage data from public web pages, we focus on capture and analysis of reader behavior with the article as a whole, rather than a specific visualization.

Other projects contribute models and tools for narrative visualization. Walny et al. [299] analyze use of visualizations through the lens of active reading. GraphScape [169] formalizes and extends earlier work by Hullman et al. [143] to suggest appropriate visualization sequences. Ellipsis [254] is a domain-specific language (DSL) and graphical interface for augmenting visualizations for use within narratives. *Idyll* (Chapter 4) is a DSL for authoring interactive articles, providing a markup language, component library, and reactive variable system to orchestrate interactive content. Vuillemot et al. [297] note a challenge for analyzing visualization usage is that it often necessary for authors to build their own logging tools to support relevant event tracking. Here, we leverage *Idyll*'s architecture—in which reactive variables directly drive interactive updates to the presented document content—to collect detailed content-specific usage logs across articles, without writing additional instrumentation code. These variables typically fully specify the state of an article, and so tracking their changes over time is sufficient to represent the interactions occurring during a reader's session.

7.2 Case Study Article Design

To better understand how readers on the web engage with interactive narrative content, we instrumented three interactive articles before they were published online. The articles covered varying topics (music theory, guitar tuning, dimensionality reduction) and targeted different readerships, but each used design techniques common to contemporary interactive articles. All articles were authored using *Idyll*. *Idyll*'s reactive runtime is powered by a single global state, an architecture that lends itself to instrumentation for easy logging and

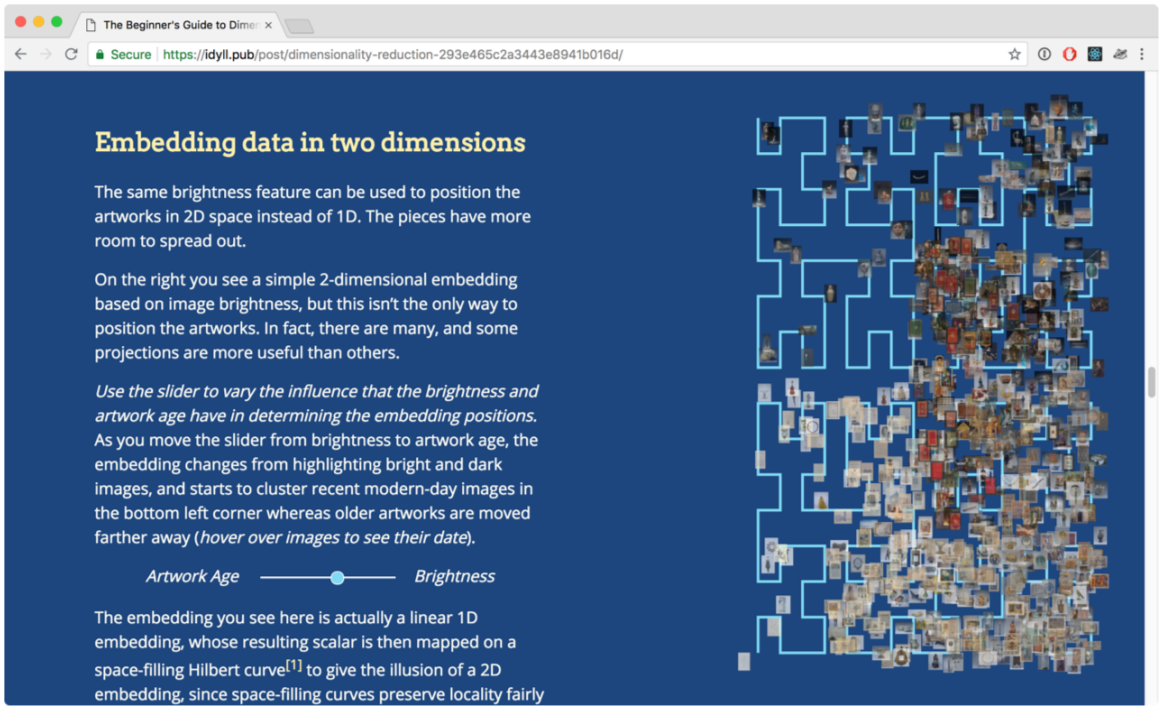


Figure 7.3: The *Beginner's Guide to Dimensionality Reduction* uses artworks from the Metropolitan Museum of Art as examples for introducing dimensionality reduction techniques.

replay, allowing us to generalize and automate the data collection process. We collected reader data on each of the articles for several weeks, however the vast majority of reader sessions came within the first two days of the articles' publication.

Here we give an overview of the article designs; in later sections, we apply our logging and analysis tools to each of the articles and discuss our findings. Each case study concerns a text-driven, linear narrative accompanied by interleaved interactive graphics and control widgets. The articles draw on a variety of design techniques that are prevalent in online articles from news outlets, for example scroll-based navigation and interactivity, inclusion of buttons that can be used to start and stop animations or audio clips, and custom interactive graphics. The articles were chosen in part because of these representative features as well as our ability to instrument them. Importantly, each article was designed by the authors to

address a topic of personal interest that they believed to be of value to a broader audience, providing an authentic and realistic motivation.

7.2.1 Beat Basics

Beat Basics (Figure 7.2) was written by a computer science undergraduate as a submission to the 2018 Explorables Jam [11], a three week online hackathon that solicited submissions of explorable explanations covering any topic. The article teaches readers about the difference between two non-standard musical time signatures ($\frac{3}{4}$ and $\frac{6}{8}$), using interactive graphics based on John Varney's rhythm wheel [287]. The article was written to be accessible to an audience without formal training in music theory, and uses audio-enabled interactive graphics to convey rhythms. There are 7 distinct sections of content. Most of the user interactions are tied to hover events: readers can move their mouse cursor over different graphics on the page to trigger audio playback and animation. There are also buttons that may be clicked in order to reveal additional information. At the end of the article users can listen to a section of a classical song and watch it synchronize with the interactive rhythm wheel.

7.2.2 The Beginner's Guide to Dimensionality Reduction

The *Beginner's Guide to Dimensionality Reduction* [85] (Figure 7.3), was created as a submission to the 2018 Workshop on Visualization for AI Explainability [1]. It was written by two computer science Ph.D. students. The article was crafted to make dimensionality reduction techniques accessible to those who did not have a pre-existing intuition for why the techniques are useful or how they work, and consists of 7 distinct sections. It starts by introducing a dataset of artworks from the Metropolitan Museum of Art [216], and uses this data in examples throughout the article. Dimensionality reduction is introduced first in one dimension, sorting the artworks by brightness, moves on to a simple but naïve 2-dimensional projection, and ends with an overview of three algorithms that are used in modern data science practice: PCA [18], t-SNE [180], and UMAP [182]. Readers can hover over an artwork to call up more details. As they scroll through different sections, the graphics

update automatically in response. Other interactions are available,including a parameter to the naïve 2-dimensional projection, a button that can be clicked to display more technical details, and buttons to reveal details about the algorithms discussed.

7.2.3 How To: Tune a Guitar

How To: Tune a Guitar (Figure 7.1) uses interactive graphics and audio to teach readers techniques for tuning guitars. The article was written by two co-authors of this paper. The topic was chosen because we expected that it would be interesting to a wide audience, and the article was written to be accessible to beginners without prior musical knowledge while still being of interest to experienced musicians. We developed an interactive guitar widget with audio and dynamic visualizations to create a fun yet educational article. The article asks readers to tune the interactive guitar, and use several different methods to achieve the goal of tuning the guitar properly. This article is the longest of the three discussed in this paper, consisting of 13 distinct sections. The interactive graphics update in response to readers scrolling through content. There are several opportunities for direct user interaction with the widgets, including strumming the guitar, modifying tuner knobs, clicking buttons to trigger audio playback, and adjusting parameters that drive specific features such as adding distortion or modifying beat frequency playback.

7.3 Tools for Capturing Reader Activity

To automate article instrumentation, we developed an extension to the *Idyll* runtime that records a number of attributes that are semantically relevant to the reader experience (i.e., beyond low-level mouse, touch, or keyboard events). When a reader first loads an article, metadata is collected about their browser, device, and screen size. A scroll event listener is added to produce a time-series of user scroll positions. A timer is used to track the duration of user visits to each article. State changes to *Idyll*'s reactive variables are recorded, allowing for reconstruction of the usage patterns in individual reader sessions. Google Firebase is used for data storage and user identification, although a different backend could be configured if desired. By default users are anonymous, but identifiers are stable: if the same browser visits the page multiple times, each of these visits will be associated with the same identifier.

The recorded variables come directly from *Idyll*'s reactive runtime state and parameterize article components, tracking changes in response to user input. This allows us to automate the construction of user models that incorporate article specific features. For example, in *How To: Tune a Guitar* a variable update is emitted each time a user mouses over a string on the guitar or adjusts a tuning knob, allowing us to reconstruct in detail how users interacted with the widget. In *Beat Basics* and *Dimensionality Reduction* a variable is updated each time a reader scrolls through one of several waypoints, triggering additional updates to the display. While it is possible for article authors to define private state in components that would not be captured in our instrumentation code, this is uncommon in practice and wasn't the case in any of the articles examined here; *Idyll*'s variables typically fully specify the state of an article's interactive components. The correspondence between *Idyll* variables and user interaction is not a coincidence particular to these articles, but follows directly from the design of the language. Our capture methods can be used directly by any *Idyll* article. We have released our logging extension as open source software and hope that it will enable further study of interactive article usage.

7.4 Tools for Analyzing Reader Activity

To analyze captured reader activity logs, we start by computing aggregate statistics about usage of various article features, then use custom visual analysis tools (*ScrollHop* and *Readuction*) to perform exploratory analysis and identify patterns of user engagement. We also work with domain experts (the article authors) in order to create high level metrics concerning article usage, and use these to analyze the usage of features specific to each of the articles. Our analysis tools are applicable to data from any interactive article with appropriate instrumentation. To gain a high-level overview, we calculate aggregate usage statistics for each article. This includes distributions of the amount of interactions with each variable, the amount of time spent on the page, and the amount of progress made through the content. We compute these distributions by aggregating log data, and generate visualizations of usage of key variables and scroll progress (available as supplementary material at `https://github.com/uwdata/papers-interactive-analytics-eurovis2019`). Since *How To: Tune a Guitar* did not implement explicit waypoints, we track scroll position at

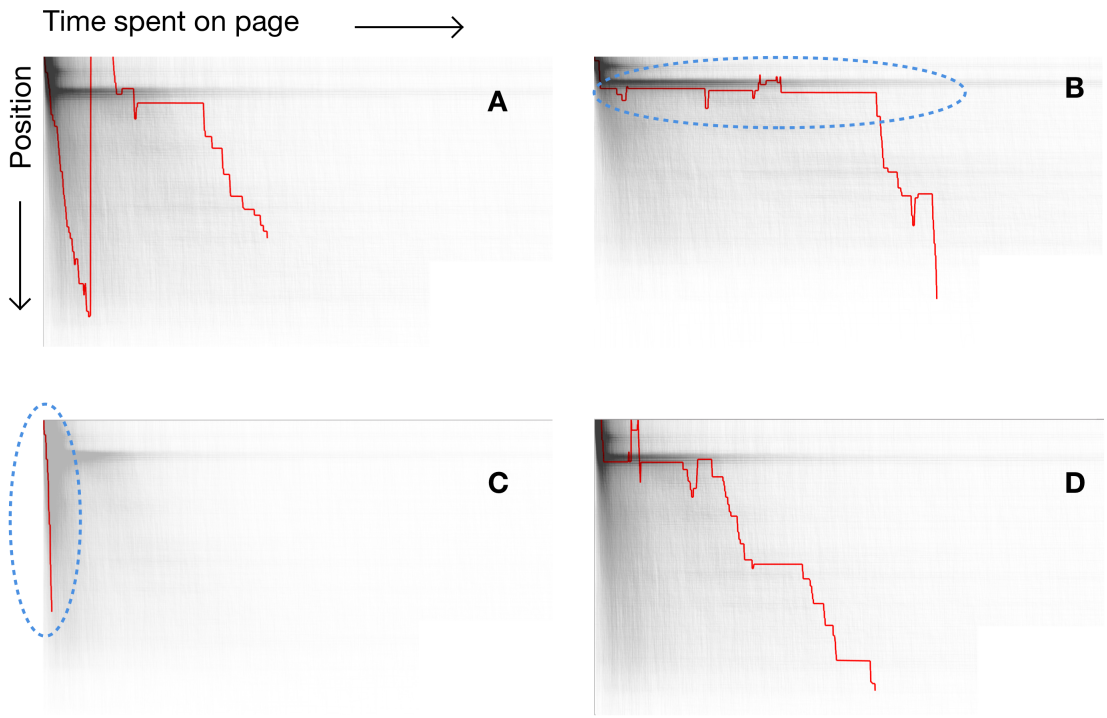


Figure 7.4: Scroll positions over time for *How To: Tune A Guitar*. Plots highlight exemplary patterns: (A) *Preview & Read*, (B) *Super Tuners*, (C) *Scroll & Bounce*, and (D) *Balanced Engagement*.

the pixel level and later bin the positions to a particular section in a post-processing step. We also asked authors of each article to formulate questions about how users might interact with their articles. We prompted authors to test their assumptions of usage as well as consider interactions that might exhibit critical engagement or content understanding. We operationalize each of these questions by writing a corresponding query over the log data and present the results among the calculated statistics.

7.4.1 ScrollHop: Visualizing Scroll Positions over Time

It is difficult to use existing techniques to identify common patterns in user scroll data. One possibility is to use dynamic time warping [47] techniques to cluster the scroll timeseries,

however the size of our data set made this computationally infeasible. The timeseries could be transformed into a series of events (e.g., navigations between article sections) and analyzed using event flow visualization, however there are challenges in using these techniques at scale [104], and we wished to be able to inspect nuanced scroll behavior within sections. To gain insight into patterns of reader behavior, we created *ScrollHop* (Figures 7.1, 7.4), a tool that visualizes each visit to the page as a single line displaying scroll position over time. Overlapping behaviors form dark bands, allowing us to assess how readers cluster in their movement through the webpage. By sequentially highlighting individual scroll timelines in red, we can see how individual scroll behaviors relate to the space of all scrolling behavior within the captured data. Prior work in uncertainty visualization [144] suggests that animating through individual outcomes is an effective strategy for understanding the variance among possible outcomes in a set. ScrollHop uses a custom WebGL shader to interactively (re-)render tens of thousands of scroll traces in response to axis scale and opacity adjustments.

Through the use of this tool we discovered interesting user behaviors such as *preview & read*, where users briefly scroll down the page to examine the content before returning to the top of the page and going back through the page more slowly. Users who *scroll & bounce* were the most prominent group, moving quickly through the webpage and exiting after a short time. The behaviors that we highlight throughout this paper map onto behaviors that we observed to be common when viewing samples drawn from the population of all user scroll patterns.

7.4.2 Readuction: Supporting Nuanced Segment Analysis

To assess more nuanced behaviors, we analyze feature vectors that summarize reader interaction with an article. Some analysis tools perform clustering to identify segments of the user population (e.g. [74]), but depending on the parameters these approaches may suffer from over- or under-clustering of groups, and can require careful evaluation [132]. As each interactive article consists of customized content and interactions, we reasoned that tools for this domain should privilege nuanced inspection and interpretation over potentially mis-

leading discrete categories. This led us to consider dimensionality reduction methods that can provide a continuous representation of similarities among reader sessions.

We define feature vectors to numerically encode reader behavior in a multi-dimensional space. The vector dimensions consist of (1) count aggregations of state changes for each *Idyll* variable, (2) the amount of time spent in each section, and (3) associated data such as the maximum scroll depth that a reader reached. Feature vectors are automatically computed from *Idyll* usage logs, with each article using a different set of dimensions depending on the *Idyll* variables employed. These variables fully specify the state of the article. The counts of their changes correspond to the overall usage of the article widgets. These features represent both the content that a reader engages with and the amount of time that they spent doing so.

We then use the feature vectors as input to the Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction algorithm [182], which uses feature similarity to form a topological mapping that is then projected into a Cartesian coordinate space. We chose the UMAP algorithm over alternatives such as t-SNE [180] or PCA [18], as we found UMAP produced useful projections that allowed us to identify different user behaviors without over-clustering, revealing smooth gradations of behavior, aligning with our design goals above. We visualize the projected 2D coordinates in a scatter plot; adjacent points represent visits which are similar in some subset of the selected features.

Readuction is an interactive tool for generating and exploring these UMAP visualizations (Figure 7.1). It displays a data point for each reader session, projected into UMAP coordinates. On the right-hand side, two bar charts show the median value of reader time spent per article section and variable usage counts. The UMAP projection and bar charts are linked so that the user can select subsets of sessions and examine the corresponding feature vectors. For each of the articles examined, we used *Readuction* to identify usage patterns which were representative of large groups (hundreds to thousands) of users. To do this, we selected different regions of reader sessions, observing the varying distributions of time spent and variable usage. We utilized the expertise of article authors to understand how these distributions map onto real-world article usage. The tool enabled us to discover cohorts of users defined by similar engagement with article content. For example, in the

analysis of *How To: Tune a Guitar* we call one such group *super tuners.*

7.5 Case Study Article Analysis

In this section we analyze each article independently using the tools and methods previously discussed. In the ensuing discussion we tie together patterns observed across the articles, highlight our conclusions, and suggest areas of future research. Because the articles were designed primarily for laptop and desktop users, we put the primary focus of the analysis on these users; throughout the analysis that follows statistics refer to desktop users, except where we explicitly refer to mobile users. Many of the article features were disabled or shown in a static format for mobile readers, so this helps us make uniform comparisons when discussing feature usage.

7.5.1 Beat Basics

After submission to the 2018 Explorables Jam, *Beat Basics* was posted online to Twitter, Hacker News, and a music education community on Reddit. We collected data from 10,000 sessions by 8,368 users, 4,456 of which viewed the desktop version of the article.

Aggregate Statistics

The median time spent on the page was 86 seconds (36-183 IQR), and 85% of readers interacted with the interactive content to playback the beats (median 16 interactions, 6-27 IQR). Of the two example songs featured at the end of the article, both the $\frac{3}{4}$ and $\frac{6}{8}$ songs were played a median of two times, indicating that these readers reached the end of the content and engaged with the audio playback mechanism for each of the time signatures multiple times. Mobile readers spent less time on the page (median 20 seconds, 6-57 IQR). In both formats, there were outliers who drove up the mean amount of time on the page (3,224 seconds desktop, 510 seconds mobile), likely due to readers leaving the page open while they performed other tasks. On mobile the median number of interactions with an audio component was one, the top 25% interacted 3 or more times. Only 10% of mobile readers triggered one of the example audio clips at the article's end.

Figure 7.5: An annotated UMAP projection of *Beat Basics* reader sessions. We used the Readuction tool to identify cohorts of readers exhibiting similar behavior by observing the distributions of feature vectors for subsets of reader sessions. The feature vectors consist of counts of variable changes (a proxy for engagement with interactive widgets) and time spent on each section of content.

We asked the article author to produce a set of questions about her audience that she would like answered, without discussing the details of our data collection process. She wanted to know:

> **Q1.** The article featured rhythm wheels positioned alongside the text, one near the top of the page and one near the bottom. *Did the differing placement of the rhythm wheels affect their usage?*
>
> **Q2.** Audio playback could be triggered by hovering over the rhythm wheels, or links in text. *How did interaction levels vary between different input modalities?*
>
> **Q3.** Readers were prompted to interact with various widgets at different points in the text. *Were readers following along and interacting in accordance with the narrative?*

To answer Q1 and Q2 we can consult the aggregate usage statistics. The distribution of usage of the $\frac{3}{4}$ circle (the one placed in the top half of the page) skews slightly higher

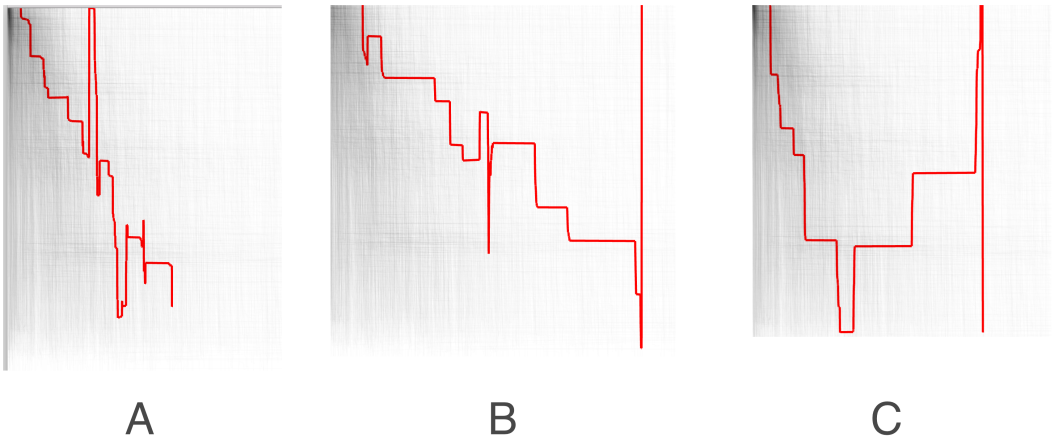


Figure 7.6: Example scroll paths from *Beat Basics*. (A) shows a reader who backtracks in the content several times; (B) a reader progresses linearly before scrolling down quickly to preview a section and subsequently spend time reading it; (C) shows a reader who read through the content forward and then in reverse.

(median 8 hovers, 4-14 IQR) than the $\frac{6}{8}$ circle (median 8 hovers, 2-12 IQR). The circles are featured more prominently and were used more than the inline links in the text. 75% of readers interacted with one of the two rhythm circles, while 54% hovered over one of the two links in the text, indicating that they seemed to be aware of the inline links, but preferred interacting directly with the larger graphic. To answer Q3, we measure the number of times that readers switched between engaging with the two different rhythms. The article used several different techniques to explain how the two rhythms compared. Readers that were following along closely with content should have switched between the two multiple times. The median number of switches was 6 (2-9 IQR), indicating that most readers were following along with the content and were completing the tasks set-up by the author in the manner that she expected. This calculation is straightforward to compute via a query over our collected interaction data, suggesting that the data collected is appropriate for answering these kinds of content-specific queries.

Observations from ScrollHop

About three quarters (73%) of *Beat Basics* readers reached the end of the article. In observing the scroll data we noticed a few trends. The first—common throughout all three articles, perhaps common to any content distributed to a broad online audience—is that many readers simply scroll through to the end of the content very quickly and leave the page. Many of these readers interact with various interactive components, although the length of time that they spend doing so suggests a very superficial level of engagement. We also noticed that some readers' trajectories were erratic, moving backwards and forwards in the article. These readers seemed to be engaging with the content as a whole rather than one individual section at a time to support their understanding of the topic, drawing on the multiple visualizations which presented the information in different ways. Figure 7.6 shows examples of these reading patterns including two individuals whose path through the content was mostly linear but included backtracking (A, B) and one who read forward through the content and then viewed it in reverse (C).

Observations from Readuction

Figure 7.5 shows both desktop and mobile readers plotted according to the UMAP dimensionality reduction algorithm; both groups are plotted using a shared projection. We used Readuction's interactive selection tool to view time and usage distributions and identify regions of interest, and then manually added annotations. Outliers who bounced from the page without interacting can be seen clustered away from the central group. Different regions of the main group can be inspected to reveal the most engaged readers, the least engaged, and the many that lie in between. We dragged the brush tool across regions of users to view aggregate feature vectors and selection counts, revealing the density and behavioral patterns exhibited in different areas of the visualization. Towards the edges of the main group are users who went furthest above or below average in utilization of different article features or time spent on the page. Throughout the plot the usage patterns of neighboring clusters of readers tended to be quite similar, indicating a continuum of engagement behavior rather than clearly separable groups.

7.5.2 The Beginner's Guide to Dimensionality Reduction

The *Beginner's Guide to Dimensionality Reduction* was submitted to the IEEE VisxAI workshop, posted to Twitter, and subsequently appeared on the Data Science community on Reddit, and on datacamp.com. The post received 7,055 visits from 4,549 readers; 3,170 sessions were on desktop devices.

Aggregate Statistics

The median time spent on the page was 116 seconds (33-386 IQR). 76% of readers reached the end of the article. Of the readers who reached the end, 59% clicked one of the buttons to select an additional algorithm. This means that 41% of the readers who "finished the article" did not read all of the content. The details of the other algorithms weren't necessary for the high-level takeaways of the articles, but they were needed for informed application of the technique in practice. The median number of times a reader hovered over an artwork was 4, the median number of interactions with the algorithm details button and with the slider was zero. Many readers quickly bounced from the page. However, many readers also engaged with the article, interacting with the various widgets and some manipulating features a great many times. Graphs of the usage distributions of key article variables are available as supplementary material at `https://github.com/uwdata/papers-interactive-analytics-eurovis2019`. The median time that mobile readers spent on the page was 40 seconds (10-131 IQR). 48% of mobile readers reached the final content of the article, and of those 38% utilized the functionality to observe details of the other algorithms. Across the articles we consistently noticed that many mobile readers would consume all of the textual content even though they did not engage with interactive features. This is likely due in part to the fact that these articles were designed with desktop readers in mind, and though responsive to some extent, did not offer a polished experience for mobile readers. It may also be due to time or environmental factors affecting mobile readers, leading them to be less likely to dig deep into features and more likely to read for the high-level takeaways.

We interviewed an author of this article to understand what features he considered most

valuable for tracking user engagement. He was interested in high level information such as completion rate and time on page, as well as granular information about how readers engaged with specific widgets. He commented that he wanted to see specific metrics for different parts of the interactive content.

> **Q4.** The article featured a slider allowing readers to adjust a parameter of a two-dimensional embedding. *Were readers engaging with this slider? If so, how were they using it?*
>
> **Q5.** At the end of the article readers could click on buttons to see details about three different dimensionality reduction algorithms. Clicking a tab caused the visualization to be redrawn using the corresponding algorithm. *Were readers examining these details? Were they engaging with the visualization in order to understand different algorithms' output?*

When discussing Q4, the author commented "If I had a distribution of slider values, I could see how people were using it. If it's bimodal maybe we should have used a button." The distribution of values taken on by the slider represents all values with approximately equal weight. However, if we look instead at the minimum value that the slider took on, shown in Figure 7.7, there is clearly a bimodal distribution. This indicates that very few readers left the slider in the middle without first moving it to the lower end of the range. The readers tended to rarely interleave usage of the slider and hovering over artworks—only ˜5% of readers did this—indicating that readers weren't spending time examining the consequences of moving the slider, and that this presentation was not as effective as intended. Overall about 45% the readers engaged with the algorithm details buttons. Only about 15% of readers used one of these buttons and subsequently engaged with the visualization.

Observations from ScrollHop

We noticed several trends when observing the scroll behavior for this article. First, the data seem to display much heavier horizontal banding than either of the other two articles, indicating that readers were spending more time pausing to read content. Because this

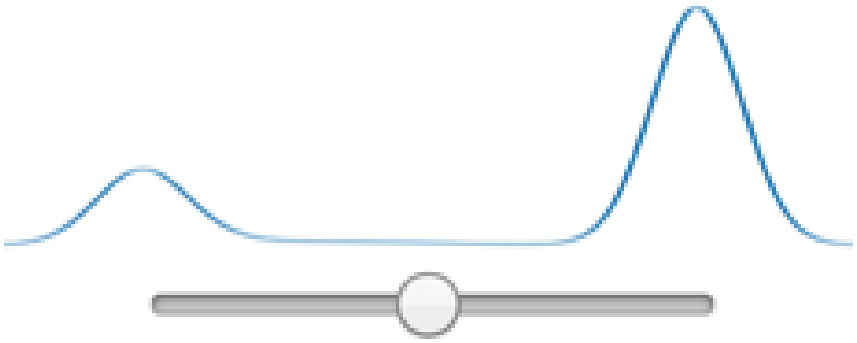


Figure 7.7: The distribution of minimum values of a range slider in *The Beginner's Guide to Dimensionality Reduction*. The slider started in the far right position. Many users did not engage with it; others moved it all the way to the left.

is both the most technical and the most didactic of the three articles this pattern is not surprising. Readers tended to scroll through in a linear manner, working their way forward through the article, seldom jumping between sections. This may imply that the content was well motivated and well matched to the audience, or it may be an artifact of our design choice to use scroll-based waypoints and animations, perhaps making the process of switching sections more burdensome than free-scrolling. As with all of the articles we noticed a *preview & read* behavior, where users would quickly scroll down the page before scrolling back up and either bouncing or continuing to read through at a slower pace. We also noticed that some readers would progress quickly downwards into the story before pausing at one of the sections and continuing to engage more heavily (Figure 7.8.A). This may indicate that the content after a certain point in the article was well matched to their background, or that the article's design piqued an interest [124]. Some readers went back and re-read sections after finishing the content (Figure 7.8.B).

Observations from Readuction

Figure 7.9 shows the UMAP projection for desktop readers of this article. Most readers cluster into one mass, with several smaller "tendrils" detached from the main group. This view was helpful in allowing us to identify subsets of users based on where they spent their

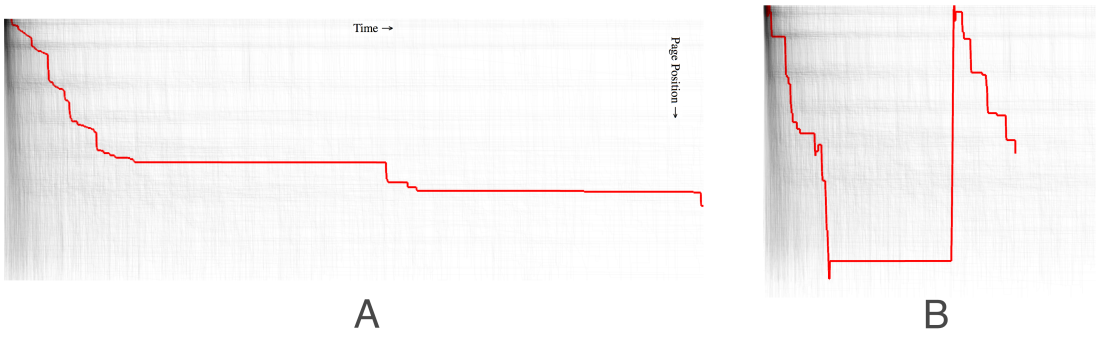


Figure 7.8: Example scroll paths from the *Beginner's Guide to Dimensionality Reduction*. (A) shows a reader who skimmed the beginning of the article, but spent more time in the later sections; (B) shows a reader who quickly reached the end of the article, spent time there, and then went back to re-read earlier content.

time and what they interacted with. Using the selection feature we further identified groups of readers who exhibit similar behavioral quirks, such as leaving the web page open for some time before actually reading the article. Users with no or low engagement are visible off the main cluster. Within the main cluster we can identify users who engaged with interactive components to varying degrees, for example, those used the buttons to click through details of all of the algorithms several times (*algorithm explorers*), those who engaged heavily with the weight parameter (*weight shifters*), and those who scrolled through the article multiple times (*re-readers*).

7.5.3 How To: Tune a Guitar

How To: Tune a Guitar was published and posted to *Twitter* and *Hacker News*, and was subsequently covered by *GIGAZINE*, a Japanese news blog, and *Codedrops Collective*, an online web design publication. It also appeared in Google search results for the search term "how to tune a guitar." We collected data from 55,193 sessions by 46,347 readers, including 27,603 desktop sessions.

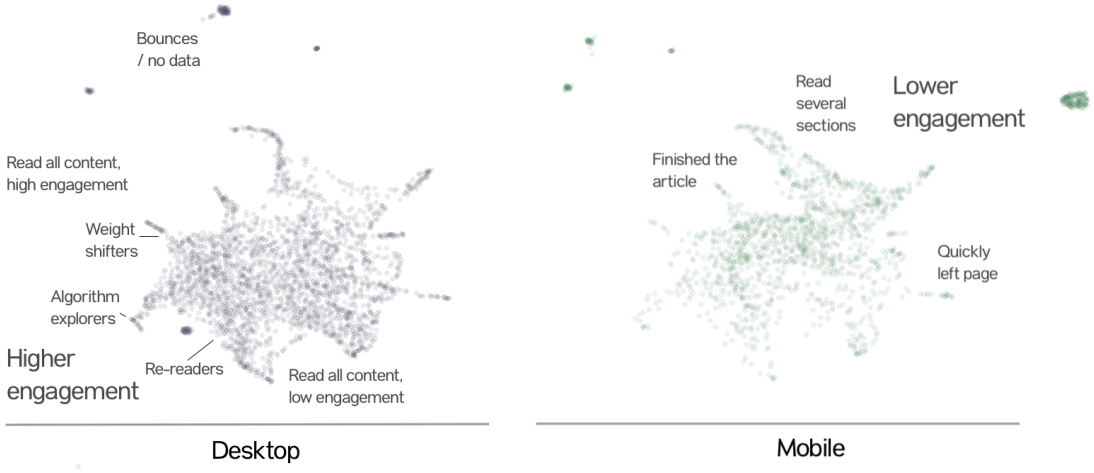


Figure 7.9: UMAP projection of feature vectors for *Beginner's Guide to Dimensionality Reduction* readers. The projection reveals broad clusters of readers, such as those who consume all the content, those who leave the page quickly, those who engage heavily with article features, and those read the content multiple times.

Aggregate Statistics

The median time spent on the page was 78 seconds (27-210 IQR), and readers performed a median of 22 "plucks" of the guitar string (6-56 IQR), and 42 tuning adjustments (6-119 IQR). 23% of readers tuned all six strings of the guitar. Usage of the more detailed interactive features was relatively low: just 14% of readers clicked the button to play an example of beat frequencies, 2% modified the beat frequency playback. For mobile readers, the median time on page was 41 seconds (10-131 IQR). 82% of mobile readers advanced through at least some of the content, even though they needed to dismiss a warning about download size; however, only 34% attempted to tune the guitar and just 6% tuned all six strings. These observations suggest that mobile users are willing to engage with interactive content, and that the specific interactions should have been refined to better accommodate them. The author-posed questions revealed additional information about reader interactions:

Q6. The article featured an interactive guitar widget that would play audio when a

reader hovered over a string. Readers could quickly move their mouse over multiple strings to "strum the guitar". *How many times did readers strum the guitar?*

Q7. The article asked readers to use tuners on the guitar widget to tune the guitar multiple times. *How many unique times did readers actually tune the guitar?*

By querying the dataset for interactions with multiple strings on the guitar in quick succession, we can answer Q6 and determine how many times readers were strumming the guitar (median 1 strum, 0-3 IQR). Readers triggered the audio playback of an individual string (median audio triggered 22 times, 6-56 IQR) more than they strummed, indicating that they tended to use the guitar for tuning (as the narrative suggested), rather than playing freely. We answer Q7 by counting the number of times that readers interact with a guitar tuner, separated by some other interaction on the page. We found that readers had a median of 1 tuning session (0-4 IQR). This suggests that most readers did interact with the tuners when prompted initially, but only the most engaged readers continued to follow along through the rest of the content, including attempting to tune by beat frequency. This content may have been too advanced for readers without musical experience, or users may have been less inclined to continue interacting after the novelty wore off.

Observations from ScrollHop

Only 45% of readers reached the final section. This completion rate is considerably lower than the other two articles; this article had the most content, it also was shared with the largest and most general audience. Figure 7.4 shows four exemplary scroll patterns. As in other articles, some users would *preview and read* (A), and some would *scroll and bounce* (C); (B) shows a reader playing with the guitar near the beginning of the article but skimming through later content; in (D) a reader exhibits balanced engagement.

Observations from Readuction

The UMAP projection (Figure 7.10) again reveals a central group of users, separated from notable outliers. Near the edges of the central cluster we can identify interesting groups,

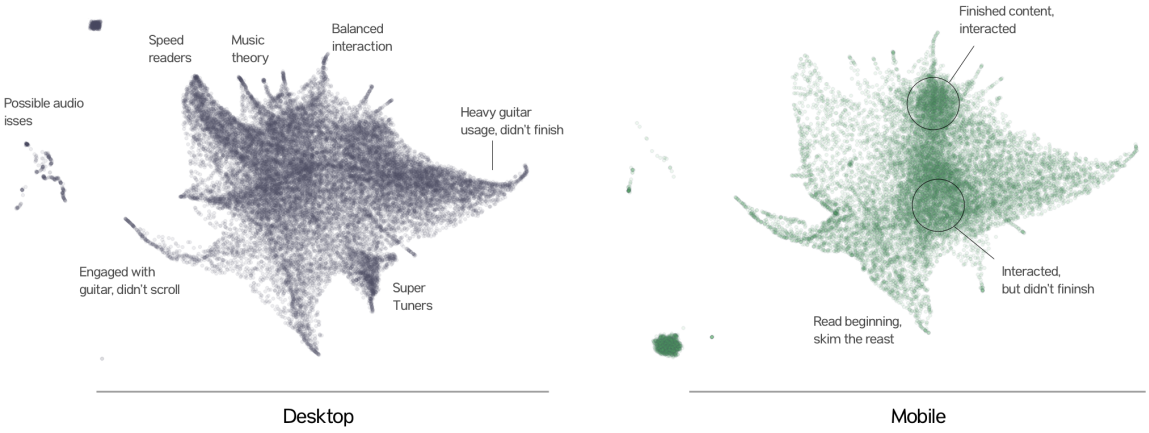


Figure 7.10: UMAP projection of feature vectors for readers of *How To: Tune a Guitar*. This article offered the most polished mobile experience of the three, and we see a less stark difference in the distributions between the desktop and mobile readers. Through interactive use of our Readuction tool we identify clusters of users who exhibit interesting engagement patterns.

such as *super tuners* who spent most of their time on the page tuning the guitar, *music theory* readers who engaged heavily with more advanced topics but skimmed through earlier sections, and *speed readers* who spent less than 10 seconds on average in each section, yet still engaged with many of the interactive components. Of the three articles, this plot shows the least obvious separation between mobile and desktop readers, likely stemming from the fact that this article offered the most polished presentation on mobile devices and readers could still engage with many of the interactive features. However, we also noticed a large group of mobile readers who engaged with interactive content but did not finish the article, possibly indicating that the novelty of the guitar interaction had worn off or that these readers were responding to other time or environmental issues. We noticed a group of desktop readers separated from the main cluster who attempted to interact with the guitar but did not appear to do anything else on the page; we believe that these users may have hit a bug related to the Chrome WebAudio API that some users had reported.

7.6 Discussion

Through our case studies of article instrumentation and analysis, we have demonstrated the feasibility of large-scale *in situ* deployment and logging to study interactive article readers, and validated the use of our proposed tools to gain insights into reader behavior. We observed many nuanced patterns of reader engagement, including some that corroborate prior research and some that are new to this work. Certain behaviors were consistent across the articles, including patterns of scrolling through content and the distributions of some aggregate usage statistics. We conclude by sharing recurring findings and implications of our techniques and analysis.

7.6.1 Recurring Reader Behaviors

We saw many readers quickly scroll through article content before leaving the page [56] or engaging in depth (*preview & read*). This suggests that readers make an initial assessment for quality and content, and aligns with prior work which found that aesthetics play a large role in the levels of user engagement [124]. We found that readers who do interact display a *continuum of behaviors* and do not necessarily fall into clearly separable groups. While many readers interacted with the articles in a manner that suggests they are *following the narrative*, some readers will engage disproportionately with various article components, likely *matching on content* that they find the most relevant. Both form and content seemed to impact behavior: in *Beat Basiscs*, which allowed for free scrolling between sections, readers would often *backtrack* to compare the behavior of different interactive audio widgets; in the *Beginner's Guide to Dimensionality Reduction*, a highly technical article, readers would progress through the article linearly, pausing in each section to consider the content. A list of all observed recurring reader behaviors is available in the supplementary material at `https://github.com/uwdata/papers-interactive-analytics-eurovis2019`.

There has been a debate in the data journalism community about the effectiveness of interaction [281, 42, 22]. One suggestion is that content central to a narrative should not be hidden behind a click [281]. In this work we similarly saw that if details were only revealed after a click, readers may never see it. In *Beginner's Guide to Dimensionality Reduction*,

roughly half of desktop readers and 38% of mobile readers who made it to the end of the article clicked to reveal more details about the projection algorithms. This may indicate that some readers were not aware of the functionality, or that they were content with more high-level information. On the other hand, we also found that readers were more likely to interact with widgets which played a central role in the narrative. This can be seen in the high usage levels of the guitar widget in *How To: Tune a Guitar*, the artworks in *Beginner's Guide to Dimensionality Reduction*, and the rhythm wheels in *Beat Basics*. This indicates that users are willing to engage with interactive content on the page, but most are focused on content that is crucial to the central narrative.

For each of the case study articles, we noticed that the time that mobile users spent with the content was lower than that of desktop users. The median time spent on page for mobile users was 23-52% of the median time of desktop users. As more articles are evaluated, it should be possible to refine these statistics to better understand how a mobile article is performing relative to a desktop one, even if we expect readers to spend less time with the content. The results from *How To: Tune a Guitar* indicate that mobile users desire to view this style of content, and the majority (86%) are willing to click through a warning note about application download size, indicating that developers should take care to consider these users, noting that they are often equal to or greater in number than desktop readers. Mobile readers utilized fewer of the available interactive features than their desktop counterparts, but still largely followed the presented narratives. This result indicates that designers should ensure that the narrative is intact and comprehensible even if a reader chooses not to engage with the interactive widgets.

7.6.2 Tooling for Capture and Analysis

To understand diverse reader behaviors, we found it critical to provide insight into both aggregate (macro) behavior and individual (micro) experiences [282]. For example, both our HopScroll and Readuction visualizations provide overviews of user behavior and reveal clusters, while also supporting inspection of the particulars of individual traces. We found the smooth alternation between these levels of detail to be indispensable. Formulating

metrics in conjunction with domain experts, on the other hand, helped to check author assumptions and key interaction patterns that might not otherwise be clear in the mass of log data. Going forward, our presented tools can be used to automatically generate interactive reports. Given author-guided queries as input, the various components described above—log analysis, aggregate statistics, feature vector calculation, and resulting interactive visualizations—could be synthesized to produce article dashboards for real-time tracking by authors and editors. The system can operate by analyzing usage of all *Idyll* variables (or a user-defined subset), and support linking between views, for example highlighting ScrollHop traces of reader sessions selected in the Readuction tool.

Further work is needed to adapt HopScroll to articles which don't rely on the browser's built-in scrolling functionality for content navigation. For example, an article which utilizes a stepper or slideshow motif may not require the reader to scroll at all. In these cases we expect that the same visualizations will be effective, so long as reader progress can be modeled as a linear progression with a corresponding progress fraction. In addition, we do not yet render desktop and mobile reader data simultaneously in HopScroll because the different article layouts lead to differences in where the content is displayed on the page and makes comparison difficult. A normalization step is needed to visualize and compare the scroll behavior of these two groups directly.

The use of *Idyll* to author these articles greatly simplified article instrumentation: in addition to scroll position, we could track all changes to *Idyll*'s state to produce relevant reader feature vectors and queries to answer author inquiries. These experiences point to the value of domain-specific languages or design tools to not only author content, but help evaluate its use. To enable more systematic study of the effects of different choices or article structure or presentation, future work might augment *Idyll* to support authoring alternative versions in parallel. Akin to A/B testing, *Idyll*'s runtime and analytics system could then assign incoming readers to article versions, enabling experimental control at the language level.

7.6.3 Implications for Future Research

A difficulty of conducting research on interactive articles "in the wild" is access to log data from a broad audience, especially when private media companies are reticent to share metrics publicly. One possibility is to use Mechanical Turk, but it is not clear that crowd workers behave in the same manner as intrinsically-motivated online readers. A nice property of interactive articles is that there is an online community who is interested in consuming this type of content. During the course of this research we took care to design articles in such a way that they would be appealing to specific audiences as entertaining and educational, regardless of the fact that they were also being used for research.

Our work demonstrates that publishing public content is a viable complement to laboratory studies and paid crowdsourcing. This approach introduces article design challenges beyond standard research design issues, but holds the promise of more comprehensive and realistic study of narrative visualization. We advocate that research goals in this domain should be pursued by studying authentic stories of genuine interest to the authors and intended readers. We hope the contributions of this chapter provide first steps down this path, and expect that controlled comparisons of article variants can be performed using these tools and methodology. To facilitate future research, our instrumentation and analysis tools are available as open-source software as part of the *Idyll* project, and the full interaction logs from our case studies are available online at `https://s3-us-west-2.amazonaws.com/interactive-analytics-eurovis/data.zip`.

Chapter 8

CONCLUSION

Interactive articles can encourage readers to actively engage with complex material and support designs that can improve learning outcomes. They are currently very time consuming to produce and require a wide range of expertise. To this point we've discussed work that surveys the design of interactive articles, authoring systems that facilitate the creation of interactive articles without use of general purpose programming tools, and tools that facilitate the collection and analysis of usage data in order to assess article effectiveness. I begin with a brief review of the contributions of this dissertation, then discuss a number of future research directions including future direction for design and authoring tools and discuss the role of automation in the interactive storytelling process, and leave with concluding remarks summarizing this line of work and dissertation.

8.1 Summary of Contributions

The work in this dissertation adds to our understanding of interactive articles and provides tools that facilitate authoring, publishing, and evaluating them. In Chapter 3, through an analysis of the design of sixty interactive articles, I identified key affordances of the medium: *Connecting People and Data*, *Making Systems Playful*, *Prompting Self-Reflection*, *Personalizing Reading*, and *Reducing Cognitive Load*. I connected these affordances to multimedia learning principles and digital journalism research and discussed how the medium allows authors to communicate complex data-driven technical information in a format that improves engagement, learning, and recall. I discussed the implications for authoring tools and highlighted where more research needs to be done.

In Chapter 4, I introduced *Idyll*, a highly expressive authoring tool that reduces the amount of code needed to create and publish interactive articles through a simple markup language, reactive document model, and standard component library. I showed how *Idyll*

reduces the amount of effort and custom code required to create interactive articles through examples and first-use results from undergraduate computer science students. *Idyll* has been used to create articles that have been read by hundreds of thousands of people, featured in design publications like FastCo Design [225], published at IEEE VIS workshops [86, 85], and served as a base upon which additional contributions were built in Chapters 5–7.

Idyll Studio (Chapter 5) is a graphical editor that allows users to author interactive articles using a visual interface utilizing direct manipulation interactions like *point-and-click* and *drag-and-drop*. Findings from an initial user study with 18 participants suggest that users enjoy using this type of WYSIWYG-style interface and that *Idyll Studio* lowered the threshold for non-experts to create interactive articles and allows experts to rapidly specify a wide range of article designs. All users could perform basic editing and composition, use datasets and variables, and specify relationships between components. Most could choreograph interactive visualizations and dynamic text, although some struggled with advanced uses requiring unstructured code editing.

Fidyll (Chapter 6), supports authoring interactive articles through a higher-level language, reducing the amount of non-narrative markup that authors must write to produce accessible and archiveable interactive articles across multiple formats. I conducted a series of formative interviews with seven domain experts to understand needs and practices around multi-platform narrative visualizations, and developed a cross-genre compiler for authoring narrative visualizations. *Fidyll* reduces the amount of non-narrative markup that authors need to write by 80-90% compared to using *Idyll* directly by synthesizing large swaths of markup, allowing authors to focus on their domain specific content rather than programming layouts.

In Chapter 7, I developed an automated process to collect high-level article usage data, developed tools to analyze and visualize this data, and published an open database of over 50,000 anonymized reading sessions. I contributed extensions to the *Idyll* markup language to automate the detailed instrumentation of interactive articles and developed corresponding visual analysis tools for inspecting reader behavior at both micro- and macro-levels. Three case studies of interactive articles were instrumented and posted online to reach broad audiences. I demonstrate the use of these tools to characterize article-specific interaction

patterns, compare behavior across desktop and mobile devices, and reveal reading patterns common across articles. The contributed findings, tools, and corpus of behavioral data can help advance and inform more comprehensive studies of narrative visualization.

8.2 Future Research Directions

While conducting the research contained in this dissertation, I identified a number of related areas where future research is necessary. In this section I discuss directions for providing authors with design guidance and recommendation, the role of automation in interactive article production, improving narrative visualization and interactive article authoring tools, and interactive articles in academic publishing.

8.2.1 Design Guidance & Recommendation for Interactive Article Authors

The usefulness of interactive articles is predicated on the assumption that these interactive articles actually facilitate communication and learning, however an article's effectiveness may be highly dependent on its design [194]. Current authoring tools don't provide authors with any feedback on the quality of their interactive article designs. Findings from empirical analysis (such as in Chapter 7), as well as basic principles of graphic design (e.g. [304, 206]) should be encoded into authoring tools to provide authors with feedback. Given a system like *Fidyll* in which authors provide a limited set of *domain-specific parameters* to specify a data story, can a system identify parameters which haven't been introduced or explained in the text of the story? Such a system might incorporate heuristics to determine what parts of the parameter space may be most interesting to readers, and alert authors if interesting parameter combinations haven't been sufficiently explored in the article or aren't possible to reach through interactive controls. As discussed in Chapter 7, it may be necessary to develop article-specific metrics in order to assess usage of specific designs; future tools could assist authors in creating article-specific A/B tests or hypothesis tests about how certain aspects of their article should be utilized, and provide notifications if usage anomalies are detected.

The challenge of providing guidance is difficult in part because there is limited empirical evaluation of the effectiveness of interactive articles: major publishers of interactive articles

like the *New York Times* are generally unwilling to share internal metrics, and laboratory studies may not generalize to real world reading trends. To help address this I presented a dataset of in-the-wild usage data in Chapter 7, as well as tools to facilitate the collection and analysis of such data, but more can be done. The *New York Times* provided one of the few available data points, stating that only a fraction of readers interact with non-static content, and suggested that designers should move away from interactivity [281]. However, in Chapter 7, I found that many readers, even those on mobile devices, are interested in utilizing interactivity when it is a core part of the article's message. This statement from the *Times* has solidified as a rule-of-thumb for designers and many choose not to utilize interactivity because of it, despite follow-up discussion that contextualizes the original point and highlights scenarios where interactivity can be beneficial [22]. This means designers are potentially choosing a suboptimal presentation of their story due to this anecdote.

8.2.2 Automation in Interactive Article Production

The suggestion of incorporating design feedback and recommendations in authoring systems begs the larger question of what the role of automation should play in interactive article production. Automation occurred in several places in this dissertation, such as in *Fidyll*, where automated text-to-speech was used to to add voice-over and captions to animated data videos, or in Chapter 7, where I automated the collection of semantically meaningful article usage data given article source code. While this direction is promising, there will inevitably be challenges incorporating automation into the interactive article design and production process [130]. As seen in Chapter 6, fully automating article design and layout may not be desirable. Design trends are constantly evolving, and as of now certain automated techniques like text-to-speech aren't sufficiently advanced to replace their manual alternatives. Automated solutions or design recommendations may be better thought about as offering constructive criticism or a starting substrate from which to work, rather than striving to creating publishable artifacts on their own.

Additional automation could be included, for example authoring tools could utilize text summarization techniques [116] to generate explanatory text at different levels of abstrac-

tion, allowing authors to publish articles that supported readers with widely varying backgrounds without the burden of writing multiple copies of article text. Projects like ScienceSimple [165] point to the role that automation can play in dynamic articles; authoring tools for interactive articles should consider incorporating functionality like this to allow authors to easily create personalized articles, articles that speak at multiple levels of abstraction [294], and designs like StretchText [211]. Tools & techniques from information visualization research that automate the transition and selection of interactive graphics should also be considered and incorporated into interactive article authoring systems where appropriate. For example, work that can automate animating between various information graphics [168] or suggest an interesting visual tour through data [169] could be incorporated into an authoring system like *Fidyll* to scaffold out initial drafts of articles and automate transitions between sections.

8.2.3 Collaboratively Authoring & Debugging Interactive Articles

I discussed the breadth of skills required to author an interactive article and have presented several tools that assist authors in the production process. However, the barrier to entry for creating interactive articles can still be lowered. In practice interactive articles are created by diverse teams of contributors [181], yet the current set of authoring tools don't acknowledge this. Tools should support *collaborative* authoring. Existing systems like Observable [215] support simultaneous editing of documents by multiple collaborators, yet systems could go farther to support authors: given the diverse set of participants in the data storytelling process, it may be appropriate to develop various interfaces which are tailored toward particular types of contributors, e.g. an editorial interface that editors use to make text changes and leave comments on article design and limits the amount of code and other implementation details shown.

There is also a need for better debugging support. Systems like *Idyll Studio* limit the need for general purpose programming tools, but some users still struggle when implementing interactive article designs and could be better supported by visual debugging tools. For example, with *Idyll Studio* when authors add interactivity to their article, they implicitly

create a graph that controls how data flows between user inputs, reactive variables, and components on the page; this graph could be visualized on-demand to assist in understanding program behavior. Existing in-situ debugging tools [140] in other authoring domains provide evidence that such functionality would be beneficial to authors.

8.2.4 Authoring Animated, Annotated Graphics

There are more opportunities for visualization authoring tools to support designs like annotated visualizations, supporting visual links between text and graphics, and supporting cinematic and motion graphics techniques like camera pan and zoom and animating layers in and out of view. For example, data journalists routinely publish highly polished chart designs that include annotations which adhere to gestalt principles of information design to guide readers through a specific visual path through a graphic. This type of visualization is crucial for data visualizations yet today is typically added to visualizations as an additional visual layer in a tool like Microsoft PowerPoint or Adobe Illustrator. Some research tools offer support for certain aspects of this—Ellipsis [254] supports adding annotations to graphics made with D3 or other JavaScript libraries to build narrative visualizations; VizFlow [275] supports linking text to annotations drawn over raster visualizations—but these tools don't link the visualizations and the annotation semantically; there is no clear pathway to for example, modify part of the underlying visualization scene graph (e.g. to highlight a region of points) when an annotation comes into view, or modify the annotation based on how a user interacts with the visualization; any logic of this type must be manually specified. Grammar of graphics [257, 306] approaches may support labels [171] or generic text channels, but don't offer first-class support for annotations. There is also a wide space of designs that have found use in data videos which are also useful for interactive articles. Shi et al. [263], offered a dseign space for this style of motion graphics as it applies to narrative visualization. There are also an increasing number of interactive article which utilize rich 3D graphics to walk readers through a physical scene or artifact, for example Reuter's *Drowning in Plastic* which utilized 3D models and concrete scales [76] to teach readers about the scale of plastic usage worldwide.

8.2.5 Academic Journals that Support Dynamic Media

Many interactive articles are self-published due to a lack of platforms that support interactive publishing. Creating more outlets that allow authors to publish interactive content will help promote their development and legitimization. The few existing examples, including newer journals such as *Distill*, academic workshops like VISxAI [1], open-source publications like *The Parametric Press* [86], and live programming notebooks like Observable [215] help, but currently target a narrow group of authors, namely those who have programming skills. Such platforms should also provide clear paths to submission, quality and editorial standards, and authoring guidelines. For example, news outlets have clear instructions for pitching written pieces, yet this is under-developed for interactive articles. Lastly, there is little funding available to support the development of interactive articles and the tools that support them. Researchers do not receive grants to communicate their work, and practitioners outside of the largest news outlets are not able to afford the time and implementation investment. Providing more funding for enabling interactive articles incentivizes their creation and can contribute to a culture where readers expect digital communications to better utilize the dynamic medium. Academics should push to accept the creation of new interactive and visual explanations of research topics as valid contributions to the field.

8.3 Concluding Remarks

In this dissertation, I contributed new tools for authoring, publishing, and evaluating interactive articles, and showed how the emerging medium helps authors communicate complex ideas more effectively and can spark a desire to learn and engage in readers. I presented a suite of tools—the foundation of which is the *Idyll* markup language—that lower the barrier to entry for article creation and publication, and support authors in collecting and analyzing usage data, supporting future empirical studies and design guidance. The software discussed in this dissertation has been used by researchers, educators, and journalists to create articles that resonated with hundreds of thousands of readers and that received coverage in popular press. The software is open-source and is available at `https://github.com/idyll-lang`.

BIBLIOGRAPHY

[1] VISxAI Workshop at IEEE VIS 2018. http://visxai.io.

[2] Jekyll. https://jekyllrb.com/, 2008.

[3] Khan Academy | Free Online Courses, Lessons & Practice. https://www.khanacademy.org/, 2008.

[4] React – A JavaScript library for building user interfaces. https://reactjs.org/, 2015.

[5] Live results: Presidential election. *The Washington Post*, 2016.

[6] Idyll download count. https://npm-stat.com/charts.html?package=idyll, 2017.

[7] Online Journalism Awards - ArchieML - 2016 Gannett Foundation Award for Technical Innovation in the Service of Digital Journalism finalist, Dec 2017.

[8] Puppeteer. https://pptr.dev/, 2017.

[9] Chartbeat. https://chartbeat.com/, 2018.

[10] Explorable explanations. https://explorabl.es/, 2018.

[11] The explorables jam! https://explorabl.es/jam/, 2018.

[12] Google analytics. https://analytics.google.com/, 2018.

[13] Heap. https://heapanalytics.com/, 2018.

[14] How does the eye work. *Explorable Explanations Game Jam*, 2018.

[15] Matomo (formerly piwik). https://matomo.org/, 2018.

[16] Call for proposals winter/spring 2019. https://parametric.press/issue-01/call-for-proposals/, 2019.

[17] Espen J Aarseth. *Cybertext: Perspectives on ergodic literature*. JHU Press, 1997.

[18] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Comput. Stat.*, 2(4):433–459, July 2010.

[19] Eytan Adar, Carolyn Gearig, Ayshwarya Balasubramanian, and Jessica Hullman. Persalog: Personalization of news article content. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3188–3200. ACM, 2017.

[20] Eytan Adar and Elsie Lee. Communicative visualizations as a learning problem. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):946–956, 2020.

[21] Mortimer J Adler and Charles Van Doren. *How to read a book: The classic guide to intelligent reading*. Simon and Schuster, 2014.

[22] Gregor Aisch. In defense of interactive graphics. `https://vis4.net/blog/2017/03/in-defense-of-interactive-graphics/`, 2017.

[23] Gregor Aisch, Amanda Cox, and Kevin Quealy. You draw it: How family income predicts children's college chances. *The New York Times*, 2015.

[24] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Christophe Hurter, and Pourang Irani. Understanding data videos: Looking at narrative visualization through the cinematography lens. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1459–1468, 2015.

[25] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Jason Leboe-McGowan, and Pourang Irani. Hooked on data videos: assessing the effect of animation and pictographs on viewer engagement. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, pages 1–9, 2018.

[26] Anders Andersen. A note on reflection in python 1.5. *Lancaster University*, 46, 1998.

[27] Jeremy Ashkenas and Alicia Parlapiano. How the recession shaped the economy, in 255 charts. web, 2014.

[28] Judith Aston and Sandra Gaudenzi. Interactive documentary: setting the field. *Studies in documentary film*, 6(2):125–139, 2012.

[29] B Atkinson. Hypercard. *Cupertino, CA: Apple Computer*, 1987.

[30] Simon Attfield, Gabriella Kazai, Mounia Lalmas, and Benjamin Piwowarski. Towards a science of user engagement (position paper). In *WSDM workshop on user modelling for Web applications*, pages 9–12, 2011.

[31] Kalid Azad. Colorized math equations. *Better Explained*, 2017.

[32] Benjamin Bach, Natalie Kerracher, Kyle Wm Hall, Sheelagh Carpendale, Jessie Kennedy, and Nathalie Henry Riche. Telling stories about dynamic networks with graph comics. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3670–3682, 2016.

[33] Benjamin Bach, Nathalie Henry Riche, Sheelagh Carpendale, and Hanspeter Pfister. The emerging genre of data comics. *IEEE computer graphics and applications*, 37(3):6–13, 2017.

[34] Benjamin Bach, Zezhong Wang, Matteo Farinella, Dave Murray-Rust, and Nathalie Henry Riche. Design patterns for data comics. In *Proceedings of the 2018 chi conference on human factors in computing systems*, pages 1–12, 2018.

[35] Sriram Karthik Badam, Andreas Mathisen, Roman Rädle, Clemens N Klokmose, and Niklas Elmqvist. Vistrates: A component model for ubiquitous analytics. *IEEE transactions on visualization and computer graphics*, 25(1):586–596, 2018.

[36] Emily Badger, Claire Cain Miller, Adam Pearce, and Kevin Quealy. Extensive data shows punishing reach of racism for black boys. *The New York Times*, 2018.

[37] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. A survey on reactive programming. *ACM Comput. Surv.*, 45(4):52:1–52:34, August 2013.

[38] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[39] Robert L Bangert-Drowns, Chen-Lin C Kulik, James A Kulik, and MaryTeresa Morgan. The instructional effect of feedback in test-like events. *Review of Educational Research*, 61(2):213–238, 1991.

[40] Anna Maria Barry-Jester, Ben Casselman, and Dana Goldstein. Should prison sentences be based on crimes that haven't been committed yet. *FiveThirtyEight*, 2015.

[41] Kayce Basques. A ui that lets readers control how much information they see. 2018.

[42] Dominikus Baur. The death of interactive infographics? – dominikus baur – medium, Mar 2017.

[43] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, 2000.

[44] Michel Beaudouin-Lafon and Wendy E Mackay. Reification, polymorphism and reuse: three principles for designing visual interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 102–109, 2000.

[45] Stephen Beckett. Click 1,000: How the pick-your-own-path episode was made. *BBC*, 2019.

[46] Weston Beecroft. On variable level-of-detail documents. 2016.

[47] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[48] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[49] Riccardo Maria Bianchi, Fred Hohman, and Matthew Conlen. On particle physics. *The Parametric Press*, 2019.

[50] David Blood, Joanna S. Kao, Nicolai Knoll, Robin Kwong, Callum Locke, and Ændrew Rininsland. The uber game. *Financial Times*, 2017.

[51] Ian Bogost. *Persuasive games: The expressive power of videogames*. Mit Press, 2010.

[52] Ian Bogost, Simon Ferrari, and Bobby Schweizer. *Newsgames: Journalism at play*. Mit Press, 2012.

[53] Michelle A Borkin, Zoya Bylinskii, Nam Wook Kim, Constance May Bainbridge, Chelsea S Yeh, Daniel Borkin, Hanspeter Pfister, and Aude Oliva. Beyond memorability: Visualization recognition and recall. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):519–528, 2015.

[54] Michelle A Borkin, Azalea A Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. What makes a visualization memorable? *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2306–2315, 2013.

[55] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and ccomputer Graphics*, 17(12):2301–2309, 2011.

[56] Jeremy Boy, Francoise Detienne, and Jean-Daniel 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, pages 1449–1458, New York, NY, USA, 2015. ACM.

[57] Jeremy Boy, Anshul Vikram Pandey, John Emerson, Margaret Satterthwaite, Oded Nov, and Enrico Bertini. Showing people behind data: Does anthropomorphizing visualizations elicit more empathy for human rights data? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5462–5474, 2017.

[58] Judd D Bradbury and Rosanna E Guadagno. Documentary narrative visualization: Features and modes of documentary film in narrative visualization. *Information Visualization*, 19(4):339–352, 2020.

[59] Zuggy Brain. SubRip.

[60] John Branch. Snow fall: The avalanche at tunnel creek. *The New York Times*, 2012.

[61] Matthew Brehmer, Bongshin Lee, Benjamin Bach, Nathalie Henry Riche, and Tamara Munzner. Timelines revisited: A design space and considerations for expressive storytelling. *Transactions on Visualization and Computer Graphics (TVCG)*, 23:2151 – 2164, September 2017.

[62] Meredith Broussard and Katherine Boss. Saving data journalism: New strategies for archiving interactive, born-digital news. *Digital Journalism*, 6(9):1206–1221, 2018.

[63] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[64] Larry Buchanan, Haeyoun Park, and Adam Pearce. You draw it: What got better or worse during obama's presidency. *The New York Times*, 2017.

[65] Bill Burdick. leisure. `https://github.com/zot/Leisure`, 2011.

[66] Jordi Cabot. Wordpress: A content management system to democratize publishing. *IEEE Software*, 35(3):89–92, 2018.

[67] Shan Carter, David Ha, Ian Johnson, and Chris Olah. Experiments in handwriting with a neural network. *Distill*, 2016.

[68] Shan Carter and Michael Nielsen. Using artificial intelligence to augment human intelligence. *Distill*, 2(12):e9, 2017.

[69] Nicky Case. How i make explorable explanations. `https://blog.ncase.me/how-i-make-an-explorable-explanation/`, 2017.

[70] Nicky Case. Loopy: a tool for thinking in systems. `https://ncase.me/loopy/`, 2017.

[71] Nicky Case. Explorable explanations: 4 more design patterns. https://blog.ncase.me/explorable-explanations-4-more-design-patterns/, 2018.

[72] Nicky Case. How to remember anything for forever-ish. https://ncase.me/remember/, 2018.

[73] Ben Casselman, Matthew Conlen, and Reuben Fischer-Baum. Gun Deaths in America. *FiveThirtyEight*, 2016.

[74] Marco Cavallo and Çagatay Demiralp. Clustrophile 2: Guided visual clustering analysis. *CoRR*, abs/1804.03048, 2018.

[75] Kerry Shih-Ping Chang and Brad A. Myers. Gneiss: spreadsheet programming using structured web service data. *Journal of Visual Languages & Computing*, 39(Supplement C):41 – 50, 2017. Special Issue on Programming and Modelling Tools.

[76] Fanny Chevalier, Romain Vuillemot, and Guia Gali. Using concrete scales: A practical framework for effective visual depiction of complex measures. *IEEE transactions on visualization and computer graphics*, 19(12):2426–2435, 2013.

[77] Michelene TH Chi. Self-explaining expository texts: The dual processes of generating inferences and repairing mental models. *Advances in Instructional Psychology*, 5:161–238, 2000.

[78] Michelene TH Chi, Miriam Bassok, Matthew W Lewis, Peter Reimann, and Robert Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13(2):145–182, 1989.

[79] Sahil Chinoy. Quiz: Let us predict whether you're a democrat or a republican. *The New York Times*, 2019.

[80] Ruth C Clark and Richard E Mayer. *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. John Wiley & Sons, 2016.

[81] Andy Coenen and Adam Pearce. Understanding umap. https://pair-code.github.io/understanding-umap/, 2019.

[82] Matthew Conlen. Using apparatus with idyll. https://mathisonian.com/writing/apparatus, 2017.

[83] Matthew Conlen and Jeffrey Heer. Idyll: A domain specific language for the rapid developmentof interactive news articles. In *Computation+Journalism Symposium*, 2017.

[84] Matthew Conlen and Jeffrey 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*, pages 977–989, 2018.

[85] Matthew Conlen and Fred Hohman. The beginner's guide to dimensionality reduction. *Workshop on Visualization for AI Explainability (VISxAI) at IEEE VIS*, 2018.

[86] Matthew Conlen and Fred Hohman. Launching the parametric press. 2019.

[87] Matthew Conlen, Fred Hohman, Victoria Uren, Sara Stalla, and Andrew Sass. Issue 01: Science & society. *The Parametric Press*, 2018.

[88] Matthew Conlen, Alex Kale, and Jeffrey Heer. Capture & analysis of active reading behaviors for interactive articles on the web. In *Computer Graphics Forum*, volume 38, pages 687–698. Wiley Online Library, 2019.

[89] Matthew Conlen, Alex Kale, and Jeffrey Heer. Capture & analysis of active reading behaviors for interactive articles on the web. *Computer Graphics Forum (Proc. EuroVis)*, 2019.

[90] Matthew Conlen, Megan Vo, Alan Tan, and Jeffrey Heer. Idyll studio: A structured editor for authoring interactive & data-driven articles. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 1–12, 2021.

[91] Diana I Cordova and Mark R Lepper. Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology*, 88(4):715, 1996.

[92] Michael Correll. Ethical dimensions of visualization research. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[93] Amanda Cox and Kevin Quealy. Disagreements, 2018.

[94] Brock Craft and Paul Cairns. Beyond guidelines: What can we learn from the visual information seeking mantra? In *Ninth International Conference on Information Visualisation (IV'05)*, pages 110–118. IEEE, 2005.

[95] CU. Phet interactive simulations. *University of Colorado Boulder*, 2002.

[96] Matthew Daniels. Human terrain. *The Pudding*, 2018.

[97] Himel Dev and Zhicheng Liu. Identifying frequent user tasks from application logs. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 263–273. ACM, 2017.

[98] Chrysanne Di Marco, Peter Bray, H Dominic Covvey, Donald D Cowan, Vic Di Ciccio, Eduard Hovy, Joan Lipa, and Cathy Yang. Authoring and generation of individualized patient education materials. In *AMIA Annual Symposium Proceedings*, volume 2006, page 195. American Medical Informatics Association, 2006.

[99] Distill. Latest articles about machine learning. `https://distill.pub/`, 2017.

[100] Morgan Dixon and James Fogarty. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 1525–1534, New York, NY, USA, 2010. Association for Computing Machinery.

[101] Morgan Dixon, James Fogarty, and Jacob Wobbrock. A general-purpose target-aware pointing enhancement using pixel-level analysis of graphical interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 3167–3176, New York, NY, USA, 2012. Association for Computing Machinery.

[102] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. Increasing the transparency of research papers with explorable multiverse analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2019.

[103] Henri F Drake, Ronald L Rivest, John Deutch, and Alan Edelman. A multi-control climate policy process for a trusted decision maker. 2020.

[104] F. Du, B. Shneiderman, C. Plaisant, S. Malik, and A. Perer. Coping with volume and variety in temporal event sequences: Strategies for sharpening analytic focus. *IEEE Transactions on Visualization & Computer Graphics*, 23(6):1636–1649, June 2017.

[105] Hugh Dubberly and Doris Mitch. The knowledge navigator. *Apple Computer, Inc*, 1987.

[106] James R Eagan, Michel Beaudouin-Lafon, and Wendy E Mackay. Cracking the cocoa nut: user interface programming at runtime. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 225–234, 2011.

[107] Conal Elliott and Paul Hudak. Functional reactive animation. In *International Conference on Functional Programming*, 1997.

[108] Douglas C Engelbart. Augmenting human intellect: A conceptual framework. *Menlo Park, CA*, 1962.

[109] Alice Feng, Shuyan Wu, Fred Hohman, Matthew Conlen, and Sara Stalla. The myth of the impartial machine. *The Parametric Press*, 2019.

[110] Leah Findlater, Alex Jansen, Kristen Shinohara, Morgan Dixon, Peter Kamb, Joshua Rakita, and Jacob O Wobbrock. Enhanced area cursors: reducing fine pointing demands for people with motor impairments. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 153–162, 2010.

[111] Marsha E Fonteyn, Benjamin Kuipers, and Susan J Grobe. A description of think aloud method and protocol analysis. *Qualitative health research*, 3(4):430–441, 1993.

[112] Ira R Forman, Nate Forman, and John Vlissides Ibm. Java reflection in action. 2004.

[113] Dede Frederick, James Mohler, Mihaela Vorvoreanu, and Ronald Glotzbach. The effects of parallax scrolling on user experience in web design. *Journal of Usability Studies*, 10(2), 2015.

[114] Wayne C Fredrick and Herbert J Walberg. Learning as a function of time. *The Journal of Educational Research*, 73(4):183–194, 1980.

[115] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12-13):910–950, 2010.

[116] Mahak Gambhir and Vishal Gupta. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47(1):1–66, 2017.

[117] Arthur Irving Gates. *Recitation as a factor in memorizing.* Number 40. Science Press, 1922.

[118] Nahum Gershon and Ward Page. What storytelling can do for information visualization. *Communications of the ACM*, 44(8):31–37, 2001.

[119] Chaim Gingold. Earth primer. 2015.

[120] Gabriel Goh. Why momentum really works. *Distill*, 2017.

[121] Russell Goldenberg and Matt Daniels. The gyllenhaal experiment. *The Pudding*, 2019.

[122] Daniel Gomes, João Miranda, and Miguel Costa. A survey on web archiving initiatives. In *International Conference on Theory and Practice of Digital Libraries*, pages 408–420. Springer, 2011.

[123] Mohamed G Gouda and Ted Herman. Adaptive programming. *IEEE Transactions on Software Engineering*, 17(9):911–921, 1991.

[124] Esther Greussing and Hajo G Boomgaarden. Simply bells and whistles? cognitive effects of visual aesthetics in digital longforms. *Digital Journalism*, 7(2):273–293, 2019.

[125] John Gruber. Markdown, 2004.

[126] Thu-Huong Ha and Nikhil Sonnad. How do you draw a circle? we analyzed 100,000 drawings to show how culture shapes our instincts. web, 2017.

[127] Neil Halloran. The fallen of world war ii. web, 2015.

[128] Vi Hart and Nicky Case. Parable of the polygons. 2016.

[129] Ed Hawkins. Climate spirals. *Climate Lab Book*, 2016.

[130] Jeffrey Heer. Agency plus automation: Designing artificial intelligence into interactive systems. *Proceedings of the National Academy of Sciences*, 116(6):1844–1850, 2019.

[131] Jeffrey Heer. Fast & accurate gaussian kernel density estimation. In *IEEE VIS Short Papers*, 2021.

[132] Jeffrey Heer and Ed H Chi. Separating the swarm: categorization methods for user sessions on the web. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 243–250. ACM, 2002.

[133] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics*, 14(6), 2008.

[134] Jeffrey Heer and George Robertson. Animated transitions in statistical data graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, 2007.

[135] Justin Helps. Primer.

[136] Brian Hempel and Ravi Chugh. Semi-automated svg programming via direct manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 379–390, 2016.

[137] Brian Hempel, Justin Lubin, Grace Lu, and Ravi Chugh. Deuce: a lightweight user interface for structured editing. In *Proceedings of the 40th International Conference on Software Engineering*, pages 654–664, 2018.

[138] Richard Koci Hernandez and Jeremy Rue. *The principles of multimedia journalism: Packaging digital news*. Routledge, 2015.

[139] Jane Hoffswell, Wilmot Li, and Zhicheng Liu. Techniques for flexible responsive visualization design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 2020.

[140] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. Augmenting code with in situ visualizations to aid program understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[141] Fred Hohman, Matthew Conlen, Jeffrey Heer, and Duen Horng Polo Chau. Communicating with interactive articles. *Distill*, 5(9):e28, 2020.

[142] Jessica Hullman and Nick Diakopoulos. Visualization rhetoric: Framing effects in narrative visualization. *IEEE transactions on visualization and computer graphics*, 17(12):2231–2240, 2011.

[143] Jessica Hullman, Steven Drucker, Nathalie Henry Riche, Bongshin Lee, Danyel Fisher, and Eytan Adar. A deeper understanding of sequence in narrative visualization. *IEEE Transactions on visualization and computer graphics*, 19(12):2406–2415, 2013.

[144] Jessica Hullman, Paul Resnick, and Eytan Adar. Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering. *PlOS One*, 10(11):e0142444, 2015.

[145] Nathan Hurst, Wilmot Li, and Kim Marriott. Review of automatic document formatting. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 99–108, 2009.

[146] Alexander Ivanov, Kurtis Danyluk, Christian Jacob, and Wesley Willett. A walk among the data. *IEEE Computer Graphics and Applications*, 39(3):19–28, 2019.

[147] Charles Jacobs, Wil Li, Evan Schrier, David Bargeron, and David Salesin. Adaptive document layout. *Communications of the ACM*, 47(8):60–66, 2004.

[148] Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. Gan lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1–11, 2018.

[149] Amit Kapoor. Visdown, 2016.

[150] Jeffrey D Karpicke and Henry L Roediger. The critical importance of retrieval for learning. *Science*, 319(5865):966–968, 2008.

[151] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *International Conference on Learning Representations*, 2018.

[152] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[153] Jun Kato, Tomoyasu Nakano, and Masataka Goto. Textalive: Integrated design environment for kinetic typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3403–3412, New York, NY, USA, 2015. ACM.

[154] Josh Katz. Who will be president. *The New York Times*, 2016.

[155] Josh Katz. You draw it: Just how bad is the drug overdose epidemic. *The New York Times*, 2017.

[156] Josh Katz and Wilson Andrews. How y'all, youse and you guys talk. *The New York Times*, 2013.

[157] Alan C Kay. A personal computer for children of all ages. In *Proceedings of the ACM Annual Conference*, 1972.

[158] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. Kitty: Sketching dynamic and interactive illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 395–405, New York, NY, USA, 2014. Association for Computing Machinery.

[159] Daniel A Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[160] Mat Kelly and Michele C Weigle. Warcreate: create wayback-consumable warc files from any webpage. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, pages 437–438, 2012.

[161] Holger M Kienle. It's about time to take javascript (more) seriously. *IEEE software*, 27(3):60–62, 2010.

[162] Juho Kim, Philip J Guo, Carrie J Cai, Shang-Wen Daniel Li, Krzysztof Z Gajos, and Robert C Miller. Data-driven interaction techniques for improving navigation of educational videos. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 563–572. ACM, 2014.

[163] NW Kim, SC Joyner, A Riegelhuth, and Y Kim. Accessible visualization: Design space, opportunities, and challenges. In *Computer Graphics Forum*, volume 40, pages 173–188. Wiley Online Library, 2021.

[164] Yea-Seul Kim, Jessica Hullman, and Maneesh Agrawala. Generating personalized spatial analogies for distances and areas. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 38–48. ACM, 2016.

[165] Yea-Seul Kim, Jessica Hullman, Matthew Burgess, and Eytan Adar. Simplescience: Lexical simplification of scientific terminology. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1066–1071, 2016.

[166] Yea-Seul Kim, Katharina Reinecke, and Jessica Hullman. Data through others' eyes: The impact of visualizing others' expectations on visualization interpretation. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):760–769, 2017.

[167] Yea-Seul Kim, Katharina Reinecke, and Jessica Hullman. Explaining the gap: Visualizing one's predictions improves recall and comprehension of data. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1375–1386, 2017.

[168] Younghoon Kim and Jeffrey Heer. Gemini2: Generating keyframe-oriented animated transitions between statistical graphics. In *IEEE VIS Short Papers*, 2021.

[169] Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer. Graphscape: A model for automated reasoning about visualization similarity and sequencing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2628–2638, 2017.

[170] Matthew G Kirschenbaum. *Track changes: A literary history of word processing.* Harvard University Press, 2016.

[171] Chanwut Kittivorawang, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Fast and flexible overlap detection for chart labeling with occupancy bitmap. In *IEEE VIS Short Papers*, 2020.

[172] Scott R Klemmer, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A Landay. Where do web sites come from?: capturing and interacting with design history. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8. ACM, 2002.

[173] Clemens N Klokmose, James R Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. Webstrates: Shareable dynamic media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 280–290, 2015.

[174] Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[175] Amy Ko, Htet Htet Aung, and Brad A Myers. Design requirements for more flexible structured editors from a study of programmers' text editing. In *CHI'05 extended abstracts on human factors in computing systems*, pages 1557–1560, 2005.

[176] Amy Ko and Brad A Myers. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 387–396, 2006.

[177] Robert Kosara and Jock Mackinlay. Storytelling: The next step for visualization. *Computer*, 46(5):44–50, 2013.

[178] Maarten Lambrechts. Rock 'n poll: Polls explained with interactive graphics. `https://web.archive.org/web/20180307013513/http://rocknpoll.graphics/`, 2016.

[179] Sam Lau and Philip J Guo. Data theater: A live programming environment for prototyping data-driven explorable explanations. In *Workshop on Live Programming (LIVE)*, 2020.

[180] Geoffrey Hinton Laurens van der Maaten. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

[181] Bongshin Lee, Nathalie Henry Riche, Petra Isenberg, and Sheelagh Carpendale. More than telling a story: Transforming data into visually shared stories. *IEEE computer graphics and applications*, 35(5):84–90, 2015.

[182] John Healy Leland McInnes. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv*, (1802.03426), 2018.

[183] Hakon Wium Lie and Bert Bos. *Cascading style sheets: designing for the Web*. Addison-Wesley Professional, 2005.

[184] Zhicheng Liu, Bernard Kerr, Mira Dontcheva, Justin Grover, Matthew Hoffman, and Alan Wilson. Coreflow: Extracting and visualizing branching patterns from event sequences. *Computer Graphics Forum*, 36(3):527–538, 2017.

[185] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018.

[186] Zhicheng Liu, Yang Wang, Mira Dontcheva, Matthew Hoffman, Seth Walker, and Alan Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):321–330, 2017.

[187] Ed Lowther, Lilly Huynh, Mark Bryson, and Steven Connor. Booze calculator: What's your drinking nationality. *BBC*, 2017.

[188] Mikola Lysenko. regl, 2016.

[189] Kwan-Liu Ma, Isaac Liao, Jennifer Frazier, Helwig Hauser, and Helen-Nicole Kostis. Scientific storytelling using visualization. *IEEE Computer Graphics and Applications*, 32(1):12–19, 2012.

[190] John MacFarlane. Pandoc: a universal document converter. *URL: http://pandoc. org*, 2013.

[191] Pattie Maes. Concepts and experiments in computational reflection. *ACM Sigplan Notices*, 22(12):147–155, 1987.

[192] Justin Matejka, Tovi Grossman, and George Fitzmaurice. Patina: Dynamic heatmaps for visualizing application usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3227–3236. ACM, 2013.

[193] Andy Matuschak and Michael A Nielsen. *Quantum country*. 2019.

[194] Richard E Mayer. Multimedia learning. In *Psychology of learning and motivation*, volume 41, pages 85–139. Elsevier, 2002.

[195] Richard E Mayer, Gayle T Dow, and Sarah Mayer. Multimedia learning in an interactive self-explaining environment: What works in the design of agent-based microworlds? *Journal of Educational Psychology*, 95(4):806, 2003.

[196] Richard E Mayer and Cheryl I Johnson. Revising the redundancy principle in multimedia learning. *Journal of Educational Psychology*, 100(2):380, 2008.

[197] Richard E Mayer, Patricia Mautone, and William Prothero. Pictorial aids for learning by doing in a multimedia geology simulation game. *Journal of Educational Psychology*, 94(1):171, 2002.

[198] Sean McKenna, N Henry Riche, Bongshin Lee, Jeremy Boy, and Miriah Meyer. Visual narrative flow: Exploring factors shaping data visualization story reading experiences. In *Computer Graphics Forum*, volume 36, pages 377–387. Wiley Online Library, 2017.

[199] Philip K McKinley, Seyed Masoud Sadjadi, Eric P Kasten, and Betty HC Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[200] Graham McNeill and S Hale. Viz-blocks: Building visualizations and documents in the browser. 2019.

[201] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[202] Leo A Meyerovich, Arjun Guha, Jacob Baskin, Gregory H Cooper, Michael Greenberg, Aleks Bromfield, and Shriram Krishnamurthi. Flapjax: a programming language for ajax applications. In *Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 1–20, 2009.

[203] Albert Michotte. La perception de la causalité.(etudes psychol.), vol. vi. 1946.

[204] Gordon Mohr, John Kunze, and Michael Stack. The warc file format 1.0 (iso 28500). 2008.

[205] Jens Monig, Yoshiki Ohshima, and John Maloney. Blocks at your fingertips: Blurring the line between blocks and text in gp. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 51–53. IEEE, 2015.

[206] Josef Müller-Brockmann. *Grid systems in graphic design*. Verlag Gerd Hatje, 1981.

[207] J Muybridge. The horse in motion. *Nature*, 25(652):605, 1882.

[208] Brad Myers, Scott E Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, 2000.

[209] Alok Mysore and Philip J Guo. Porta: Profiling software tutorials using operating-system-wide activity tracing. 2018.

[210] NASA. Global temperature. *NASA Global Climate Change*, 2020.

[211] Ted Nelson. Stretchtext – hypertext note #8. *Project Xanadu*, 1967.

[212] Theodor H Nelson. Getting it out of our system. *Information Retrieval: A Critical Review*, pages 191–210, 1967.

[213] Francis Nguyen, Yea-Seul Kim, Joe Germuska, and Jessica Hullman. They draw it! *Midwest Uncertainty Collective and The Knight Lab.*, 2019.

[214] Michael A Nielsen. *Neural networks and deep learning*. Determination Press, 2015.

[215] Observable. `https://observablehq.com/`, 2018.

[216] The Metropolitan Museum of Art. The metropolitan museum of art open access. *Github*, 2017.

[217] OHCHR. Report on the role of digital access providers. *United Nations Human Rights Office of the High Commissioner*, 2017.

[218] Yoshiki Ohshima, Aran Lunzer, Bert Freudenberg, and Ted Kaehler. Kscript and ksworld: A time-aware and mostly declarative language and interactive gui framework. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 117–134, 2013.

[219] Chris Olah and Shan Carter. Research debt. *Distill*, 2017.

[220] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[221] Dan R. Olsen, Scott E. Hudson, Thom Verratti, Jeremy M. Heiner, and Matt Phelps. Implementing interface attachments based on surface representations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, page 191–198, New York, NY, USA, 1999. Association for Computing Machinery.

[222] Dan R Olsen Jr. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 251–258, 2007.

[223] Yuliya Parshina-Kottas, Bedel Saget, Karthik Patanjali, Or Fleisher, and Gabriel Gianordoli. This 3-d simulation shows why social distancing is so important. *New York Times*, 2020.

[224] Amit Patel. `https://twitter.com/redblobgames/status/1168520452634865665`, 2019. publisher: Twitter.

[225] John Pavlus. The secret life of a JPEG. *Fast Company*, May 2019.

[226] Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in science & engineering*, 9(3):21–29, 2007.

[227] Jeffrey M Perkel. Why jupyter is data scientists' computational notebook of choice. *Nature*, 563(7732):145–147, 2018.

[228] Tomas Petricek. Coeffects: Context-aware programming languages. 2017.

[229] Andrew Phippen, L Sheppard, and Steven Furnell. A practical evaluation of web analytics. *Internet Research*, 14(4):284–293, 2004.

[230] Polymer project. `https://www.polymer-project.org/`, 2017.

[231] Keith T Poole. *Spatial models of parliamentary voting*. Cambridge University Press, 2005.

[232] Keith T Poole and Howard Rosenthal. A spatial model for legislative roll call analysis. *American journal of political science*, pages 357–384, 1985.

[233] N Popovich and A Pearce. It's not your imagination. summers are getting hotter. *The New York Times*, 2017.

[234] Nadja Popvich, Blacki Migliozzi, Rumsey Taylor, Josh Williams, and Derek Watkins. How much hotter is your hometown than when you were born. *The New York Times*, 2018.

[235] Victor Powell. Image kernels. *Setosa*, 2015.

[236] Kevin Quealy, Robert Gebeloff, and Rumsey Taylor. Are you rich? this income-rank quiz might change how you see yourself. *The New York Times*, 2019.

[237] Vincent Quint and Irene Vatton. Grif: An interactive system for structured document manipulation. In *Text Processing and Document Manipulation, Proceedings of the International Conference*, pages 200–312, 1986.

[238] National Public Radio. dailygraphics. `https://github.com/nprapps/dailygraphics`, 2016.

[239] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R Eagan, and Clemens N Klokmose. Codestrates: Literate computing with webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 715–725, 2017.

[240] Katharina Rall, Margaret L Satterthwaite, Anshul Vikram Pandey, John Emerson, Jeremy Boy, Oded Nov, and Enrico Bertini. Data visualization for human rights advocacy. *Journal of Human Rights Practice*, 8(2):171–197, 2016.

[241] Tom Randall and Blacki Migliozzi. Earth's relentless warming sets a brutal new record in 2017. *Bloomberg*, 2018.

[242] Casey Reas and Ben Fry. *Processing: a programming handbook for visual designers and artists*. Mit Press, 2007.

[243] Donghao Ren, Bongshin Lee, and Matthew Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE transactions on visualization and computer graphics*, 25(1):789–799, 2018.

[244] Donghao Ren, Bongshin Lee, and Tobias Höllerer. Stardust: Accessible and transparent gpu support for information visualization rendering. In *Computer Graphics Forum*, volume 36, pages 179–188. Wiley Online Library, 2017.

[245] Nathalie Henry Riche, Christophe Hurter, Nicholas Diakopoulos, and Sheelagh Carpendale. *Data-driven Storytelling*. CRC Press, 2018.

[246] Kerry Rodden, Hilary Hutchinson, and Xin Fu. Measuring the user experience on a large scale: user-centered metrics for web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2395–2398. ACM, 2010.

[247] Henry L Roediger III and Jeffrey D Karpicke. The power of testing memory: Basic research and implications for educational practice. *Perspectives on Psychological Science*, 1(3):181–210, 2006.

[248] Hans Rosling. Hans rosling shows the best stats you've ever seen. 2001.

[249] Eric Roston and Blacki Migliozzi. What's really warming the world. *Bloomberg*, 2015.

[250] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[251] Grant Sanderson. 3blue1brown.

[252] Grant Sanderson and Ben Eater. Visualizing quaternions: An explorable video series. 2018.

[253] Julie Sarama and Douglas H Clements. "concrete" computer manipulatives in mathematics education. *Child Development Perspectives*, 3(3):145–150, 2009.

[254] Arvind Satyanarayan and Jeffrey Heer. Authoring narrative visualizations with ellipsis. In *Computer Graphics Forum*, volume 33, pages 361–370. Wiley Online Library, 2014.

[255] Arvind Satyanarayan and Jeffrey Heer. Lyra: An interactive visualization design environment. In *Computer Graphics Forum*, volume 33, pages 351–360. Wiley Online Library, 2014.

[256] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics*, 26(1):461–471, 2019.

[257] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2017.

[258] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2016.

[259] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. Declarative interaction design for data visualization. In *ACM User Interface Software & Technology (UIST)*, 2014.

[260] Toby Schachman and Joshua Horowitz. Apparatus. `https://github.com/cdglabs`, 2016.

[261] Pascal Schneiders. What remains in mind? effectiveness and efficiency of explainers at conveying information. *Media and Communication*, 8(1):218–231, 2020.

[262] Edward Segel and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE transactions on visualization and computer graphics*, 16(6):1139–1148, 2010.

[263] Yang Shi, Xingyu Lan, Jingwen Li, Zhaorui Li, and Nan Cao. Communicating with motion: A design space for animated visual narratives in data videos. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[264] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343. IEEE, 1996.

[265] Nate Silver. Who will win the presidency. `https://projects.fivethirtyeight.com/2016-election-forecast/`, June 2016.

[266] J Hunter Sizemore and Jichen Zhu. Interactive non-fiction: Towards a new approach for storytelling in digital journalism. In *International Conference on Interactive Digital Storytelling*, pages 313–316. Springer, 2011.

[267] Sarah Slobin. What if the data visualization is actually people. *Source*, 2014.

[268] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788*, 2017.

[269] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, et al. Tensorflow.js: Machine learning for the web and beyond. *arXiv preprint arXiv:1901.05350*, 2019.

[270] Brian Smith. Reflection and semantics in a procedural language. *Technical Report, TR-272, MIT Laboratory for Computer Science*, 1982.

[271] Kurt Squire. Video games and learning. *Teaching and Participatory Culture in the Digital Age*, 2011.

[272] Harry Stevens. Why outbreaks like coronavirus spread exponentially, and how to 'flatten the curve'. *The Washington Post*, 2020.

[273] Charles D Stolper, Bongshin Lee, N Henry Riche, and John Stasko. Emerging and recurring data-driven storytelling techniques: Analysis of a curated collection of recent stories. *Microsoft Research, Washington, USA*, 2016.

[274] Michael Strickland, Archie Tse, Matthew Ericson, and Tom Giratikanon. Archie markup language (archieml). `http://archieml.org/`, 2015.

[275] Nicole Sultanum, Fanny Chevalier, Zoya Bylinskii, and Zhicheng Liu. Leveraging text-chart links to support authoring of data-driven articles with vizflow. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2021.

[276] John Sweller. Cognitive load theory. In *Psychology of learning and motivation*, volume 55, pages 37–76. Elsevier, 2011.

[277] Tim Teitelbaum and Thomas Reps. The cornell program synthesizer: a syntax-directed programming environment. *Communications of the ACM*, 24(9):563–573, 1981.

[278] Frank Thomas, Ollie Johnston, and Frank Thomas. *The illusion of life: Disney animation*. Hyperion New York, 1995.

[279] Tampa Bay Times. lede. `https://github.com/tbtimes/lede`, 2016.

[280] The New York Times. kyt. `https://github.com/NYTimes/kyt`, 2017.

[281] Archie Tse. Why we are doing fewer interactives. *Malofiej Infographics World Summit*, 2016.

[282] Edward R Tufte. *Envisioning information*. Graphics press Cheshire, CT, 1990.

[283] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: Can it facilitate? *International Journal of Human-computer Studies*, 57(4):247–262, 2002.

[284] UI. Plato. *University of Illinois*, 1960.

[285] Eunjoon Um, Jan L Plass, Elizabeth O Hayward, and Bruce D Homer. Emotional design in multimedia learning. *Journal of Educational Psychology*, 104(2):485, 2012.

[286] Andries Van Dam. Post-wimp user interfaces. *Communications of the ACM*, 40(2):63–67, 1997.

[287] John Varney. A different way to visualize rhythm. `https://ed.ted.com/lessons/a-different-way-to-visualize-rhythm-john-varney`, 2014.

[288] Olga Vasileva. Wikipedia preview card. *Wikipedia*, 2018. https://blog.wikimedia.org/2018/04/20/why-it-took-a-long-time-to-build-that-tiny-link-preview-on-wikipedia/.

[289] Lea Verou, Amy X. Zhang, and David R. Karger. Mavo: Creating interactive data-driven web applications by authoring html. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 483–496, New York, NY, USA, 2016. ACM.

[290] Bret Victor. Magic ink: Information software and the graphical interface. `http://worrydream.com/MagicInk/`, 2006.

[291] Bret Victor. Explorable explanations. `http://worrydream.com/ExplorableExplanations/`, 2011.

[292] Bret Victor. Scientific communication as sequential art. `http://worrydream.com/ScientificCommunicationAsSequentialArt/`, 2011.

[293] Bret Victor. Tangle: a javascript library for reactive documents. `http://worrydream.com/Tangle/`, 2011.

[294] Bret Victor. Up and down the ladder of abstraction. `http://worrydream.com/LadderOfAbstraction/`, 2011.

[295] Bret Victor. Drawing dynamic visualizations. `https://vimeo.com/66085662`, 2013.

[296] Maria Virvou, George Katsionis, and Konstantinos Manos. Combining software games with education: Evaluation of its educational effectiveness. *Journal of Educational Technology & Society*, 8(2):54–65, 2005.

[297] Romain Vuillemot, Jeremy Boy, Aurélien Tabard, Charles Perin, and Jean-Daniel Fekete. Livvil: Logging interactive visualizations and visualizing interaction logs. In *Workshop IEEE VIS 2016*, 2016.

[298] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980.

[299] Jagoda Walny, Samuel Huron, Charles Perin, Tiffany Wun, Richard Pusch, and Sheelagh Carpendale. Active reading of visualizations. *IEEE transactions on visualization and computer graphics*, 24(1):770–780, 2018.

[300] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–30, 2019.

[301] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

[302] Barron Webster. Designing (and learning from) a teachable machine. *Google Design*, 2018.

[303] Smallbore Webworks and Dennis Crothers. Cutthroat capitalism: The game. *Wired*, 2009.

[304] Max Wertheimer. Laws of organization in perceptual forms. *A source book of Gestalt Psychology*, 1, 1923.

[305] Mark Whitehouse and Mira Rojanasakul. Find out if your job will be automated. *Bloomberg*, 2017.

[306] Hadley Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011.

[307] Stephen Wolfram. *Mathematica: a system for doing mathematics by computer*. Addison Wesley Longman Publishing Co., Inc., 1991.

[308] Takashi Yamamiya, Alessandro Warth, and Ted Kaehler. Active essays on the web. In *2009 Seventh International Conference on Creating, Connecting and Collaborating through Computing*, pages 3–10. IEEE, 2009.

[309] Nathan Yau. How you will die. `https://flowingdata.com/2016/01/19/how-you-will-die/`, 2016.

[310] Stephanie Yee and Tony Chu. A visual introduction to machine learning. *R2D3*, 2015.

[311] Anders Ynnerman, Jonas Löwgren, and Lena Tibell. Exploranation: A new science communication paradigm. *IEEE computer graphics and applications*, 38(3):13–20, 2018.

[312] Polle Zellweger, Bay-Wei Chang, and Jock D Mackinlay. Fluid links for informed and incremental link transitions. 1998.

[313] Polle T Zellweger, Anne Mangen, and Paula Newman. Reading and writing fluid hypertext narratives. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia*, pages 45–54. ACM, 2002.

[314] Qiyu Zhi, Alvitta Ottley, and Ronald Metoyer. Linking and layout: Exploring the integration of text and visualization in storytelling. In *Computer Graphics Forum*, volume 38, pages 675–685. Wiley Online Library, 2019.

[315] Richard Ziegfeld. Interactive fiction: A new literary genre? *New Literary History*, 20(2):341–372, 1989.

[316] Jonathan Zong, Dhiraj Barnwal, Rupayan Neogy, and Arvind Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[317] Douglas Zongker. *Creating animation for presentations*. PhD thesis, University of Washington, 2003.