Data Animator: Authoring Expressive Animated Data Graphics

John Thompson
Georgia Institute of Technology
Atlanta, GA, United States
jrthompson@gatech.edu

Zhicheng Liu
University of Maryland
College Park, MD, United States
leozcliu@umd.edu

John Stasko
Georgia Institute of Technology
Atlanta, GA, United States
stasko@gatech.edu

ABSTRACT

Animation helps viewers follow transitions in data graphics. When authoring animations that incorporate data, designers must carefully coordinate the behaviors of visual objects such as entering, exiting, merging and splitting, and specify the temporal rhythms of transition through staging and staggering. We present Data Animator, a system for authoring animated data graphics without programming. Data Animator leverages the Data Illustrator framework to analyze and match objects between two static visualizations, and generates automated transitions by default. Designers have the flexibility to interpret and adjust the matching results through a visual interface. Data Animator also supports the division of a complex animation into stages through hierarchical keyframes, and uses data attributes to stagger the start time and vary the speed of animating objects through a novel timeline interface. We validate Data Animator's expressiveness via a gallery of examples, and evaluate its usability in a re-creation study with designers.

CCS CONCEPTS

• **Human-centered computing → Visualization systems and tools**.

KEYWORDS

animated data graphics, authoring, keyframing, object matching, staging, staggering

ACM Reference Format:
John Thompson, Zhicheng Liu, and John Stasko. 2021. Data Animator: Authoring Expressive Animated Data Graphics. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3411764.3445747

1 INTRODUCTION

Animated data graphics are an increasingly popular medium for data-driven narratives on media outlets and digital publications such as The New York Times[1] and The Pudding[2]. These digital narratives present visualizations of data to help communicate information to the viewer. An increasing number include animated transitions between the visualizations. Although animation is not always beneficial for analytic tasks [12, 51], carefully designed animation of charts can facilitate detection of objects entering into a scene [7], enhance understanding and engagement through staged transitions [31, 38], and help track changes in data [20, 29].

Designing effective animated data graphics often requires thoughtful considerations on how to *coordinate the behavior of visual objects* and *pace the temporal rhythms*. For example, animated transitions can be localized in a specific component (e.g., an axis changes from linear to logarithmic scale), or happen between two completely different visual states (e.g., changing from a scatter plot to a stacked bar chart). In the former case, displacing the axis ticks and labels and updating the objects' positions might be sufficient; in the latter case with more drastic changes, designers need to decide which visual objects should enter or exit the scene, and which objects need to be merged, transformed or split.

In addition to coordinating the behavior of visual objects, the temporal pacing of animation also requires deliberation. *Staging* and *staggering* are two commonly used techniques to pace animated transition. Staging divides a complex animation into a sequence of simpler sub-transitions called stages. Staggering applies an incremental offset to the starting time of each moving object, thus avoiding the confusion caused by all the objects moving simultaneously. These techniques may be manually designed but are often driven by data. For example, staging may be performed based on the data hierarchy, while the parents animate first, and the children later [29]; the trigger time and the speed at which each shape animate can be specified as a function of data values in an attribute.

Existing animation authoring tools, however, lack support for coordinating visual objects and specifying data-driven temporal pacing. Prevalent authoring paradigms for animated graphics include [57]:

- *keyframe animation*: specify properties of graphical objects at certain points of time by setting a set of keyframes, frames in between two keyframes are generated by tweening.
- *procedural animation*: generate animation of large number of animated objects with a set of behavior parameters.
- *presets & templates*: apply predefined animation effects and configurations to objects.

Through our previous ideation study [57], we found that designers have an overall preference for keyframe animation. Under this paradigm, to perform visual object coordination, it is necessary to track and manipulate objects across different visual states or frames. It is easy to do so in traditional keyframing tools such as Adobe After Effects for animated graphics, where the number of objects is typically small. Animated data graphics, however, often contain hundreds or more visual objects, so analyzing and tracking the objects become a challenge. Selectively applying different behaviors

[1] http://www.nytimes.com/
[2] https://pudding.cool/

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445747

to different groups of objects for coordination is also non-trivial. In order to specify and control temporal rhythms, designers need to articulate how data attributes may drive the staggering of objects or map to the speed of transition. No existing keyframing tools support these tasks with ease and precision. A few template-based tools provide predetermined temporal designs, but the range of expressivity is limited.

To date, programming remains the only viable way to create expressive animated data graphics. For example, D^3 [9] leverages the Document Object Model (DOM) for interaction and graphics rendering, and provides the enter, update, exit paradigm to manage the differences between current and next visual state of a data graphic. D^3 also provides a transition management module for interpolating any objects' properties en masse, while supporting coordination of temporal aspects of animation (e.g., delay, duration, easing). However, programming takes significant time to learn, write, and preview designs. Designers often favor familiar design tools for animation and prototyping that provide expressivity and rapid feedback.

In this paper, we propose Data Animator, a system for authoring animated data graphics without programming. Our approach is rooted in the keyframe authoring paradigm and incorporates elements from the procedural and presets & templates paradigms when necessary. Animated transitions are specified between static data graphics – called *vis boards*. Data Animator adopts a dual view design (Storyboard and Timeline views) that lets users switch between levels of authoring granularity.

In order to support visual object coordination and data-driven temporal pacing, the system needs to understand the structure and properties of static visualizations. To this end, we build Data Animator on top of the Data Illustrator framework [42], which describes the visual properties, data bindings and organizational structure of objects in a static visualization.

To facilitate coordinating visual objects, Data Animator uses a matching algorithm to analyze the relationships between objects across vis boards. Animation is created by tweening the matched objects and applying animation effects (e.g., "Fade In", "Fade Out") to the unmatched entering or exiting objects. This approach supports rapid generation of animations through automation. Since automated matching may fail in certain cases, Data Animator provides a novel user interface for designers to visualize and interpret the results of the matching algorithm, and to override defaults and manually adjust transitions.

To support the authoring of data-driven temporal pacing, Data Animator supports staggering by data where the value of a data attribute determines the delay of the animating objects. We also introduce a concept called *hierarchical keyframes*, where the allocated duration for a transition cascades as a linear function of the parents' duration. Hierarchical keyframes allow the creation of expressive pacing by combining staging, staggering and grouping of graphical objects. We present multiple scenarios to illustrate this concept and discuss the design of a novel interface for specifying and visualizing hierarchical keyframes.

Finally, we demonstrate the Data Animator system's expressive capabilities via a gallery of examples and evaluate its usability from a re-creation study with 8 designers. All the participants could complete each of the six animation re-creation tasks within a few minutes. Users found Data Animator useful for supporting the daunting task of coordinating numerous objects and provided feedback on how to improve the system interface further.

2 RELATED WORK

2.1 Effects of Animation on Data Graphics

The proliferation of animation in data narratives hints to animation's benefits for communicating data. It seems a reasonable assumption that if a static visualization can help viewers understand data then a moving visualization should be even better [23]. But what specifically about animation helps to benefit data graphics? Where is the line drawn between animation that clarifies and enriches versus that which obfuscates and distracts? We seek to support designers in creating rich, compelling, and clear animations rather than animating for animation's sake. To that end, we summarize the current body of knowledge on the advantages and disadvantages of animation for data graphics.

Recent research empirically explores pacing techniques for visualization. Previous findings from Heer and Robertson demonstrate the efficacy of staging animations to help viewers track objects and estimate changing values [31]. Fisher's synopsis provides further support for staging as a logical approach to assist viewers in understanding the intermediate steps of complex transitions [23]. More recently staging has been shown to improve subjects' accuracy to comprehend transitions that aggregate data [38] and changes to dynamic networks [16].

In contrast to positive benefits for staging, Chevalier et al. cast doubt on the effectiveness of staggering. They provide evidence that staggering has a negligible, or even negative, impact on multiple object tracking tasks [15]. This could be a result of a loss of common fate (i.e., when objects move at the same velocity along parallel trajectories) as each object moves after the other object stops.

Chalbi et al. reveal that graphical perception is improved by common fate. This evidence indicates that staggering with an overlap could mitigate the common fate loss as animations are grouped rather than animate in isolation. Related data storytelling tools support staggering as it is believed to prevent occlusion and crowding in cases of structured data and certain motion paths [5, 11]. However, further evidence is needed to prove the benefits of staggering.

Trajectory bundling is a related method to group similar animations together and remove occlusion – evidence shows that this method is particularly effective when tracking multiple targets [20]. Wang et al. introduce a vector field design approach to non-linear trajectories in animated transitions that avoids scatter plot crowding [61]. Our approach implements linear trajectories for animated transitions, future improvements should consider these more advanced forms of interpolation. When considering which easing function to use, Dragicevic et al. determined that slow-in/slow-out typically outperforms other functions for point clouds yet more research is needed for other visualization types [19].

Additional findings from empirical research and systems research have produced guidelines for when and how to use animation for visualization. Tverskey et al. use the congruence and apprehension principles as guidance for assessing whether or not animation can facilitate similar comprehension and communication as static graphic representations [59]. They conclude that animation

provides no benefit for communicating complex systems. However, they make an exception for animated transitions in cases where the transition can be "accurately perceived and appropriately conceived" by viewers. Heer and Robertson build upon the congruence and apprehension principles with actionable guidelines for animation in statistical data graphics [31]. Their guidelines include: "maintain valid data graphics during transitions", "group similar transitions", "minimize occlusion", "use simple transitions", and "use staging for complex transitions." Informed by the design and implementation of a timeline storytelling tool, Brehmer et al. propose guidelines for animated transitions between timelines of varying representation, scale, and layout [11]. They encourage the use of staging in cases of highly salient changes, staggering of object groups during translation and scaling transitions, and simple transitions that involve at most one dimension change at a time.

2.2 Visualization Grammars and Toolkits for Animation

Programming toolkits and visualization grammars have led the way in animation for visualization. Imperative programming toolkits [14, 47] update graphics in a stepwise manner on each frame update. Our approach is more closely related to declarative programming approach in which graphics animate between declared visual states without worrying about the minutia of drawing each frame. Declarative grammars provide further abstraction from the low-level details needed to create interactive visualizations [54, 55]. Similar to how the graphics rendering of Data Animator leverages the GPU for performance, Ren et al. also provide access to GPU-enhanced rendering in a familiar declarative programming library for producing visualizations [50]. gganimate [41], an extension of ggplot2 [62], builds on a foundation grammar [63] with its own "animation grammar" that tweens different components of the data to graphic pipeline (e.g., data, aesthetic mappings, coordinate systems). This approach focuses on animating a single, isolated data graphic, rather than creating animated transitions between two visualizations. More recently, Tong et al. introduced a high-level domain-specific language (DSL) that enables declarative specifications of chart animations by leveraging data-enriched SVG charts [24]. The Canis language supports applying animation effects and temporal functions to selected marks or groups of marks. However, programmers must rely on carefully formatting data-enriched SVG files in order to coordinate matching objects between each file. Kim and Heer [39] introduce a declarative grammar for specifying transition steps and build a recommender to assist designers. One of our goals is to help designers define transition steps and additional pacing methods through a graphical user interface, and automated recommendation of animation design is not within the scope of our work.

D^3 [9] is ubiquitously used to create animated data narratives by data journalists, bloggers, and designers. Leveraging the Document Object Model (DOM) for interaction and graphics rendering, D^3 provides a flexible approach for programming visually diverse data presentations. D^3's "transition" module animates selected DOM elements from their initial visual properties to newly declared visual properties. Beyond animated transitions, D^3 also offers modules that assist in state-based animations that are typically reserved for imperative programming languages (e.g., force-directed simulations). This broad range of animation would not be possible without D^3's enter, update, exit paradigm to manage the differences between current and next visual state of a data graphic. We model our approach for matching objects between visualizations after the enter, update, exit paradigm. In addition, D^3 supports pacing selected elements with declarative functions to vary delay, duration, and easing based on data. However, D^3 still poses a challenge to designers, as they must write textual code instead of designing visually. Writing, compiling, and re-running textual code is time-consuming and difficult to learn for animation designers. This workflow contrasts sharply with how designers typically preview animations in design tools with full playback functionality. Designers often favor familiar design tools for animation that provide design freedom, direct manipulation, and rapid feedback.

2.3 Template-based Authoring Systems for Animated Data Graphics

When it comes to incorporating animation into data visualizations, the number of interactive authoring systems are few, far between and out-paced by the generative capabilities of programming toolkits. All the interactive systems that support animation are template-based [17, 18, 27, 37]: designers choose a visualization from a limited set of designs, and further add or customize through a dialog interface. In contrast, Data Animator would be first to introduce animation into a visual builder approach. Amini et al. introduced DataClips, a template system for creating data videos by sequencing clips – clips are combinations of visualization type × animation type [5]. Unlike Data Animator, DataClips does not support animated transitions between two visualizations. The Microsoft platform supports animation by coupling charts and tables created in Microsoft Excel to the animation effects of PowerPoint [43]. Additional domain-specific tools support animated transitions between timeline visualizations [10, 11] and node-link graph layouts [56]; in Data Animator, we try to design and build a general-purpose animation authoring tool.

Most related to our approach, Flourish Studio allows users to create animated data narratives by sequencing visualizations in a slideshow interface [37]. Flourish Studio also supports templated "data update" animations such as bar chart races, zoomable hierarchies, and globe geo-connections. However, these systems trade off expediency for expressivity, as they help designers quickly create animated data stories yet restrict designs to a predetermined lexicon. The template approach addresses the complexities of animated data graphics by constraining the problem. With limited visualization types, coordinating behavior of visual objects between a transition can be pre-programmed; while support for pacing temporal rhythms is set to inflexible defaults. Our approach seeks to address these considerations in a broader design space for expressive animated data graphics.

2.4 (Non-data) Animation Design Tools

The prevalent authoring paradigms in animation design tools are: keyframe animation, presets & templates, and procedural animation [57]. We base our approach on the keyframe animation paradigm. In keyframe animation properties of objects are declared at

certain points in time, the differences between a set of keyframes is tweened to create intermediary frames. Previous interactive systems adopt presets & templates as they reduce the time and effort needed to create animations. While our approach is not based on presets & templates, we selectively introduce templates to hide the low-level details for complex functions such as data-driven pacing. Finally, our approach does not support procedural animation – as the imperative model of procedural animation is at odds with the declarative structure of keyframing. While direct manipulation interfaces have been proven to be effective in animating procedural illustrations [35, 36, 60], they are ill-suited for transitioning between two visual states and can produce unexpected results.

Our approach aims to augment familiar user interfaces and interactions from existing animation design tools. The timeline editor is common in keyframe animation interfaces such as Adobe After Effects [2], Invision [33] and TumultHype [58]. Keyframes' temporal positions are directly manipulated through click-and-drag interactions. Timelines are often tightly coupled with composition previews that allow scrubbing the playhead of the timeline to rapidly inspect animations. Our approach augments the traditional timeline editor for manipulating the keyframes for groups and hierarchies of objects to support data-driven pacing and relates to the Repeater [34] and Blend [44] plugins for Adobe After Effects which provide mechanisms to generate shapes and control timing en mass. Prototyping tools such as Adobe Xd [3] and Figma [22], and presentation software such as Keynote [6] and Microsoft Powerpoint [43] have recently introduced storyboarding as a way to create keyframes. In storyboarding interfaces, users define transitions between screens, views, or slides by matching objects between them. However, these tools lack the ability visualize and disambiguate object matches between views – we introduce a novel interface for data-driven matching. Preset animation effects are used in presentation tools [6, 43] to inject animation into an otherwise static object, we borrow a handful of these preset effects such as "Fade Out" and "Fade In." Presentation tools also forgo timeline interfaces for ordering the times of groups and objects with "animate after" or "animate with" conditional statements. Our approach forgoes conditional timing yet manages to introduce pacing with a hierarchical approach: keyframes are specified as a percentage of the available time allotted by parent objects (e.g., groups, collections, stages, transition).

3 DATA ANIMATOR: DESIGN OBJECTIVES AND OVERVIEW

In this section, we first introduce design objectives that inform the conceptual framework and user interfaces of Data Animator. We then discuss what kind of knowledge an animation authoring system should assume about static visualizations and describe our choice of the input graphics format. Finally, we briefly summarize the Data Animator system and introduce an example data narrative about urbanization in East Asia [13].

3.1 Design Objectives

The design objectives (**DO**) target an authoring system for users with a design background and minimal programming experience. We derive the following objectives from our prior research on how designers conceptually approach authoring animated data graphics [57].

DO1: Focus on keyframe animation yet introduce additional authoring paradigms when advantageous. Results from [57] indicate that designers prefer keyframe animation because it is the paradigm they are most familiar with. The participants also noted that combining multiple paradigms to strike a balance between ease of use and control. In Data Animator, we use keyframing as the primary authoring paradigm, where animating from one data graphic to another is achieved by treating the source and destination data graphics as two keyframes and then tweening the differences between them. We also try to identify scenarios where keyframing may become tedious, and introduce additional paradigms in these cases.

DO2: Augment familiar design concepts from existing animation tools. Similar to **DO1**, our objective is to leverage concepts from existing graphic, animation, and prototyping tools. This approach can improve the system's learnability by re-using concepts and user interfaces that we reasonably assume designers to be familiar with. When necessary, we augment these concepts to support data graphics and aim to interweave them within a fluid authoring experience.

DO3: Promote automated yet flexible matching of objects in a transition. One of the challenges in animating data graphics is to specify the matching between objects' in the source data graphic and those in the destination data graphic. We aim to design an automated matching method to reduce the burden on designers. Since a fully automatic approach may not be perfect, it is also important to clearly present the matching results to the designers and provide them the flexibility to fix errors and make adjustments.

DO4: Compose relative timing components to author expressive pacing techniques. Authoring meaningful animations in data graphics requires support for pacing techniques such as staging, staggering, or speed variation. These techniques prescribe that the triggering and duration of an object's animation depends on the animating behavior of another object. For example, staging breaks down a transition into sub-transitions that start animating after the previous animation stage ends. These relative timing components often are determined by the attributes of data bound to the objects, or the hierarchical relationships between the objects in the visualization. It is our goal to bridge the gulfs of execution and evaluation [32, 45] in authoring these timing components.

3.2 Assumptions about Static Visualizations

We consider the authoring of *static* visualizations out of the scope of this work, because it is a well-studied area with many tools available to use [42, 48, 49, 52]. A prerequisite for using Data Animator is thus to prepare static visualizations that can be used as keyframes. A variety of formats for static visualizations are available, such as raster images, scalable vector graphics (SVG), and other proprietary formats used by different authoring tools. The format of a static visualization has significant implications on how it needs to be imported and processed. For example, a raster image contains no information about the marks, their properties and data bindings,

and would require sophisticated computer vision techniques to extract such information.

Since our focus is on authoring animated transitions, we need a format the describes the following pieces of information about a static visualization:

- **Marks**: properties of all the graphical objects in a visualization including shape, geometry, and visual styles such as fill color, stroke width, stroke color, and opacity.
- **Hierarchical Organization**: in a visualization, marks are often grouped to form higher-level constructs (e.g., a glyph may consist of multiple marks [53]; multiple marks may form a collection (e.g., stacked bar chart). The format should describe such hierarchical relationships between graphical objects clearly.
- **Peers**: given selected a mark or a glyph, we want to find its peers, which refer to all the other instances that are generated together with it. For example, in a scatter plot, the peers of any circle are the other circles representing the same type of data items. Being able to get the peers of an object is important because when a visualization consists of multi-class marks, it is easy to separate these classes.
- **Data Scopes**: for each graphical object (low-level marks or high-level constructs) that represents data, the format should describe what data rows are bound to the object as its data scope [42, 53].
- **Visual Encodings**: for each visual property that encodes data, the format should specify which data attribute is encoded.
- **Scales and Axes**: for each visual encoding, the format should record the associated scale information, including scale type (e.g., linear, logarithmic), the domain and the range. Axes or legends are graphical manifestations of scales. It is desirable to have their information such as positions on screen, because we also want to support animated transitions of axes and legends.

Based on these requirements, we examined a few visualization formats. The SVG format records information about marks and sometimes hierarchical organization, but extra efforts are needed to infer data scopes and visual encodings [30]. The dSVG approach used in Canis [24] requires users to manually define if two objects from different static visualization are representing the same data through the ID and datum tags, but visual encoding or scale information is not included.

We eventually decided to choose the Data Illustrator framework [42] and its associated file format as our input visualization specification. Starting with a mark, Data Illustrator uses the *repeat* or *partition* operators to generate its **peer** marks and bind data rows to each mark as its **data scope**. Repeat and partition can be concatenated multiple times to create **hierarchical structures**. Users can also bind data attributes to visual properties, and the system automatically generates **scales and axes**. Throughout the authoring process, Data Illustrator records all the required information in its static visualization output. For a full description of the Data Illustrator framework, refer to the original publication [42]. Data Illustrator uses the JSON file format to represent all the information. The supplemental materials contain sample Data Illustrator files

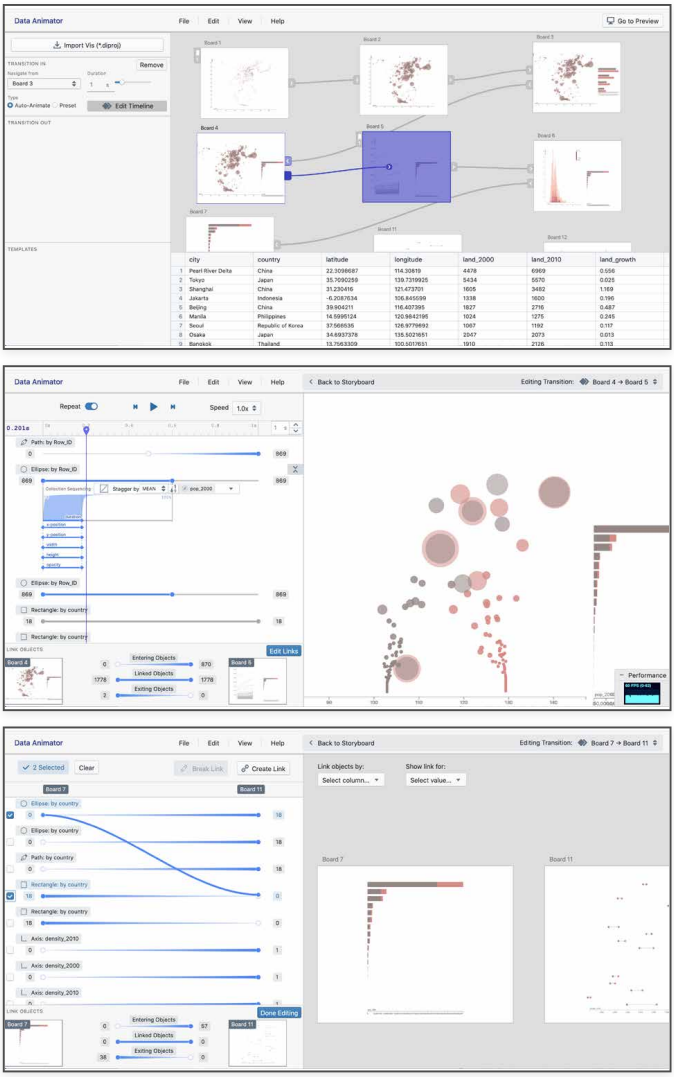


Figure 1: The main interfaces of Data Animator: Storyboard View (top), Timeline Editor (middle), Object Matcher (bottom).

that can be imported as vis boards, and they are using the ".diproj" extension.

3.3 Overview

Data Animator allows users to create animated data graphics from sequences of visually diverse data graphics. The tool consists of three views for authoring animated transitions at different levels of granularity:

- **Storyboard View** (Figure 1 - top): Here users import static visualizations created in Data Illustrator as *vis boards*, which are considered as keyframes. Users can sequence the vis boards in a 2D workspace. This design is inspired by storyboarding, a technique applied in film production and UX

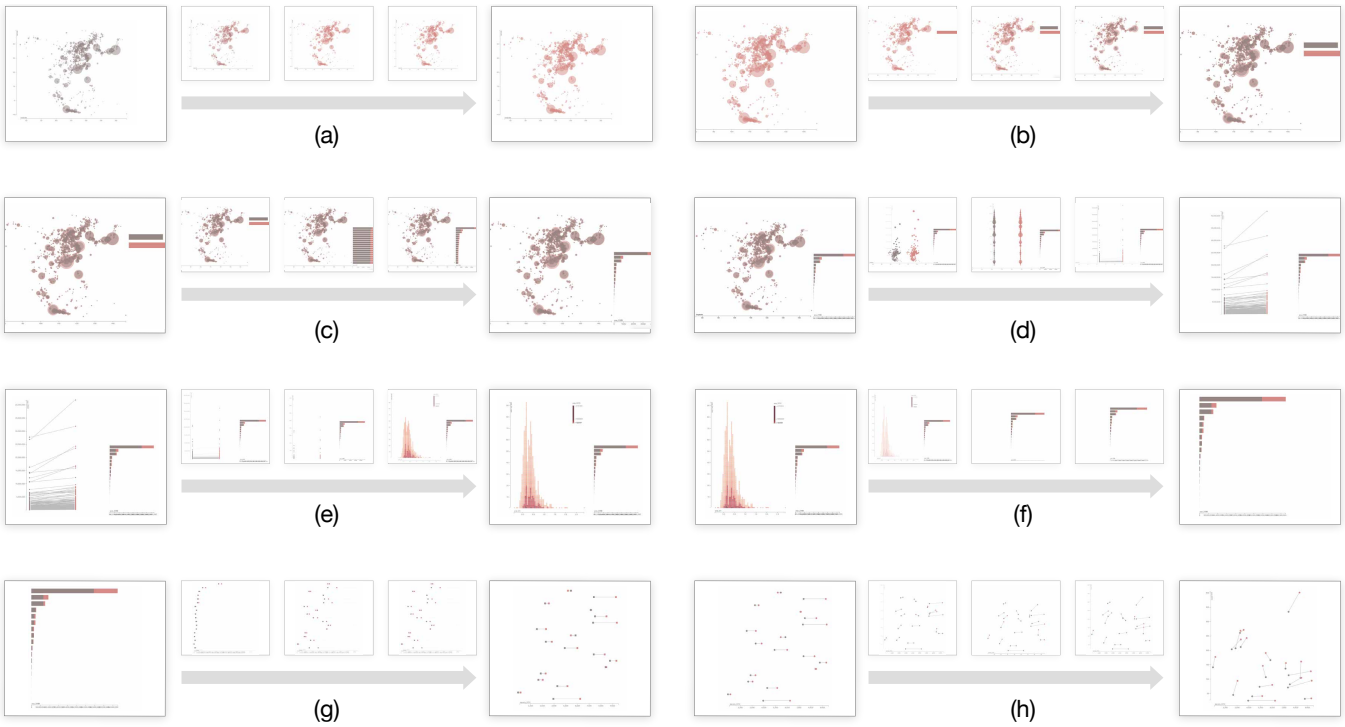


Figure 2: Still frames of featured animated transitions from "Urbanization" example scenario. (a) Symbol map of population for 2000 transition to population for 2010; (b) Introduce grouped bar chart by economic type to right; (c) Change bar chart to compare by country; (d) Symbol map transitions to slope chart for population growth from 2000 to 2010; (e) Circles of slope chart to histogram for % population growth; (f) Exiting histogram to scale in summary bar chart; (g) Rectangles of bar chart change to circles of dumbbell plot; (h) Introduce y-axis to transition to connected scatter-plot.

prototyping [3, 8, 22, 33]. *Vis boards* can be arranged, duplicated, or removed from the project. To create an animated transition, the user clicks and drags a board's connector to then drop on a desired *destination vis board*. Data Animator automatically generates a transition based on an object matching algorithm. All object timings are set to defaults that can be overridden in the "Timeline Editor".

- **Timeline Editor** (Figure 1 - middle): Clicking on any connection between two vis boards in the Storyboard View will switch to a Timeline Editor, where users can view how objects across two vis boards are matched through an automated algorithm. They can also manipulate the timelines to pace temporal rhythm of animating objects.
- **Object Matcher** (Figure 1 - bottom): If the matching of visual objects does not align with users' expectation, they can invoke the Object Matcher View through a button and tweak the automated results of object matching.

In the next two sections, we explain the working mechanisms of Data Animator in detail. We will use a data narrative about the urban population growth in East Asia from 2000 to 2010 (referred to as "Urbanization") as a running example. We base this example on the award-winning project by Nadieh Bremer [13]. The dataset contains 869 rows for each urban city in eastern Asia, with twelve columns including geographic details (e.g., `city`, `country`, `latitude`, `longitude`) and population growth quantities (e.g., `2000 population`, `2010 population`). The animated data graphics in this example transition through nine unique visualizations – Figure 2 features all eight animated transitions. The story deftly moves from maps of each urban city (Figure 2.a) to introduce auxiliary bar charts (Figure 2.b-c) to slope charts (Figure 2.d) to histograms (Figure 2.e) then zooms in to bar charts that compare growth by country (Figure 2.f), and finally drives home how economic growth of each country is tied to urbanization with a connected scatter-plot (Figure 2.g-h). The video illustrating all of these animations is at http://data-animator.com/gallery/urbanization.html.

Section 4 addresses the challenge of coordinating visual objects between transitioning data graphics by detailing the object matching algorithm of the framework. We then describe the Object Matcher, a novel interface for users to visualize and manually disambiguate the matching between two data graphics. Section 5 details how Data Animator empowers designers to pace the temporal rhythms of a transition through the Timeline Editor.

4 COORDINATING OBJECTS IN TRANSITION

To generate an animated transition from a *source vis board* to a *destination vis board*, Data Animator needs to know how the objects (e.g., marks, glyphs, collections, axes & legends) in these two vis boards should map to each other. Even though we have all the

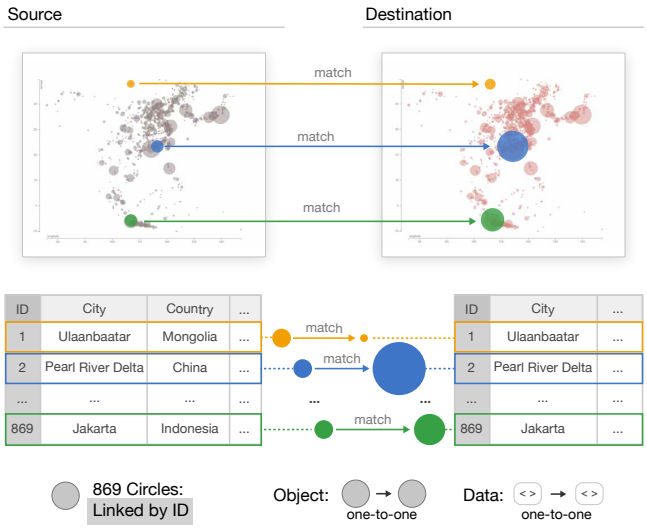


Figure 3: Example of one-to-one matching from transition in Figure 2.a.

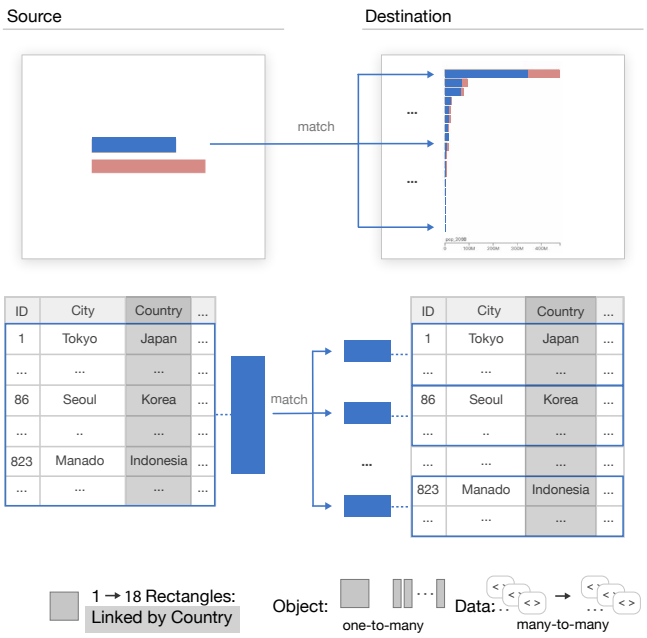


Figure 4: Example of one-to-many matching from transition in Figure 2.c. The data scope of the blue bar in the source vis board is the union of the data scopes of all the blue bars in the destination vis board.

information about graphical objects and data bindings in a single static visualization, automatically matching the objects across two vis boards is non-trivial.

In the simplest case, there is a one-to-one mapping between the objects in the two vis boards. In Figure 3, each circle in the *source* vis board (left) matches to an area-encoded circle in the *destination* (right) when they have the same data scope, that is, both are bound to the same data rows with the same city value and the same country value. During the animation, the differing visual property values of matched objects are interpolated to form a transition – for example in Figure 3 the `area` of the matched circles change in the animated transition. The matching applies not only to marks but also other visual object types such as axes & legends.

There is not always a one-to-one mapping between objects in the source and destination vis boards, however. Objects can merge or split between vis boards. For example, in Figure 4, there are two bars in the source vis board, representing year 2000 and 2010 respectively. In the destination vis board, we have two superimposed bar charts, where each row represents a country, and the two bars in each row represent the two years 2000 and 2010. The data scope of the blue bar in the source includes all the data rows where the year is 2000. In the destination vis board, the data scope of each blue bar is a data row where the year is 2010 for a specific country. The union of the data scopes of all the blue bars in the destination thus equals to the data scope of the blue bar in the source. In the animated transition, it thus makes sense to create a one-to-many matching, where the blue bar in the source splits into multiple blue bars in the destination. In such cases where a split or a merge is required, a preset effect is applied to specify that the aggregated shape *repeats* (seen in Figure 4) or *partitions* to make up the counterpart shapes in the other *vis board*.

In other cases, some objects in a vis board may simply not have any matching counterparts in the other vis board. These unmatched objects need to enter or exit the scene, and they can be animated based on a preset effect such as "Fade In" or "Fade Out" (the default presets). Preset effects create a desired animation by supplanting an object's property value (e.g., `opacity=[0]` for "Fade In").

4.1 Automated Object Matching

To handle these different cases, our overall approach of matching objects operates at the level of *object sets* instead of individual objects, and computes a *matching score* between two object sets based on a number of predefined criteria.

Given a source vis board and a destination vis board, we first group all the objects into *object sets* for each of the boards. An object set refers to a set of peer marks, peer groups or peer collections. For example, in a scatter plot, all the marks form an object set; in a trellis view of bar charts, at the highest level, we have an object set of bar charts, within each bar chart, we have an object set of rectangles. The object sets are identified through the peer information (represented as *class ID*) and the hierarchical structures recorded in a Data Illustrator file. For axes and legends, each axis or legend is an object set.

After identifying the object sets, we first check the types of members for every pair of object sets, one from the source vis board, the other from the destination vis board. By definition, the members of an object set are peers of each other and have the same type. For marks, the types can be rectangle, ellipse, path, or text. For collections, the type is either a repeat grid (members are created using Data Illustrator's repeat operator) or a partition (members are created using Data Illustrator's partition operator). For axes, the type refers to the data attribute type (categorical, quantitative or

Table 1: Components of a Matching Score between two object sets S_1 and S_2.

Component	Explanation	Scoring Function		Weight
cardinality	the number of members in the object set	$\frac{abs(\|S_1\|-\|S_2\|)}{max(\|S_1\|,\|S_2\|}$	*let S_1 and S_2 be two object sets*	3
populating field value	the categorical attribute used in repeat or partition	$\frac{abs(\|P(S_1)\|-\|P(S_2)\|)}{min(\|P(S_1)\|,\|P(S_2)\|)}$	*let P(S) be the union of all field values in object set S*	2
data scope	data rows bound to each member of the object set	$\frac{abs(\|D(S_1)\|-\|D(S_2)\|)}{min(\|D(S_1)\|,\|D(S_2)\|)}$	*let $D(S)$ be the data scope of object set S*	2
shape ID	unique identifier assigned to each mark	$\frac{abs(\|U(S_1)\|-\|U(S_2)\|)}{min(\|U(S_1)\|,\|U(S_2)\|)}$	*let $U(S)$ be the union of shape IDs in object set S*	1.5
class ID	identifier assigned to each mark to identify its peers	$\frac{abs(\|C(S_1)\|-\|C(S_2)\|)}{min(\|C(S_1)\|,\|C(S_2)\|)}$	*let $C(S)$ be the union of class IDs in object set S*	1.5

temporal). For legends, the type refers to the visual property (e.g., size, continuous color, categorical color). If two object sets have different types of members, we determine that these two object sets do not match. Otherwise, we proceed to the next step.

For a pair of object sets sharing the same type of members, we compute a matching score between them. The matching score is the weighted sum of the following components, summarized in Table 1.

- **cardinality**: the number of members in an object set. Two matching objects need not have the same cardinality. For example, the destination vis board may remove some marks in an object set from the source vis board in order to achieve an animated filtering effect; or a shape may be splitting into multiple shapes. Instead of using a binary scoring function, we compute the score of cardinality as the percent difference between the cardinalities of two object sets.
- **populating field value**: when an object set is a collection (i.e., a repeat grid or a partition), Data Illustrator records which field or data attribute was used to perform the repeat or partition operation to populate its members. For example, in Figure 3, the circles in the scatter plot were created by repeating an initial circle using the ID attribute. Each of the circles created is assigned a populating field (ID) and value (an ID value). The matching score for populating field & value is computed first by doing a union on the field & values for all the members in an object set, then calculating the percent difference between the two object sets.
- **data scope**: as mentioned in Section 3.2, data scope refers to the data rows bound to a graphical object. For example, in Figure 3, we have an object set consisting of the circles in the scatter plot. Each circle in the scatter plot in the source vis board has a data scope of exactly one row. The data scope of the object set, then, is the union of the data scopes of all the circles. The matching score for data scope is computed as the percent difference between the data scopes of two object sets.
- **shape ID**: when the vis board was created, Data Illustrator automatically assigns a unique ID to each shape. These IDs can accurately reflect how the same Data Illustrator file evolves over time, if the designer saves different states of the design as multiple vis boards over the course of authoring.
- **class ID**: id from Data Illustrator that is used for all peer shapes, this information helps us know if marks or collections are a part of same object set. The matching score is computed in the same way as the shape ID.

The overall matching score between two object sets is the weighted sum of all the above component scores. Table 1 shows the weights we assign to each component. These weights are designed based on the following considerations: cardinality, populating field value and data scope are three most indicative criteria to determine if two object sets are representing the same data. They are thus assigned the most weight. Shape ID and class ID are less reliable, for example, if designers worked on the source and destination vis boards in different sessions, the IDs may not match. They are thus assigned less weight. For a given object set in the source vis board, we pick the object set from the destination vis board with the highest matching score. If that maximum score is greater than or equal to 5 (out of 10), we designate this pair of object sets to be matching. Otherwise, there is no matching object set from the destination vis board.

We use a threshold of 5 to determine if there is a match based on a few plausible matching scenarios: whenever the cardinality and the data scope scores are both 1, or the cardinality and the populating field value scores are both 1, or the populating field value, the data scope and shape ID scores are all 1, we can say with high confidence there is a match. In all these cases, the weighted sum would be greater than or equal to 5 (out of 10).

4.2 Visualizing Object Matching Results

Data Animator enables designers to view the matching results in the Timeline Editor. As mentioned in Section 3.3, automated matching is performed when users connect two vis boards in the Storyboard View (Figure 1 - top). To view the matching results between two vis boards, users select the connection linking these boards, and click the "Edit Timeline" button. The interface displays the Timeline Editor (Figure 1 - middle). The left panel (Figure 5) shows the results of automated object matching for the transition in Figure 2(a). In this transition, the source vis board shows a bubble plot of the populations of different cities in 2000, and the destination vis board

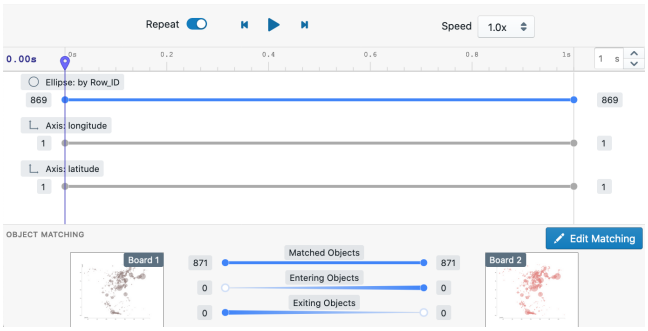


Figure 5: The left panel of the Timeline Editor, which shows the results from the automated object matching for the two vis boards in Figure 2(a). The first timeline specifies the behavior of 869 ellipses and the final two correspond to the x and y axes.

shows a bubble plot of the populations of different cities in 2010. In both visualizations, the x axis is the longitude of the cities, and the y axis is the latitude of the cities. The bottom part of the panel shows a preview of these two vis boards. The main area of the panel displays multiple *layers*, one for each pair of matched object sets. For example, the ellipses in the source and destination vis boards match up perfectly: the cardinality of the ellipse object set is 869. Such a perfect matching is represented as a timeline with the same thickness at both ends. Since the visual properties (size and fill color) of the matched ellipses are different in the two vis boards, the timeline is colored blue, indicating tweening is needed. In this case where automated object matching is successful, the animated transition as a result of interpolating the ellipses works without any user intervention. The longitude and latitude axes match up perfectly too. In addition, there are no differences in their visual properties. Their timelines are thus colored gray, indicating that no tweening is needed.

For the transition in Figure 6 (corresponding to Figure 2(e)), more object sets are involved. In the source vis board, we have a slope graph showing the changes in urban population from 2000 to 2010 (each line is a city, and the left anchor points represent year 2000, the right anchor points represent year 2010), and two bar charts

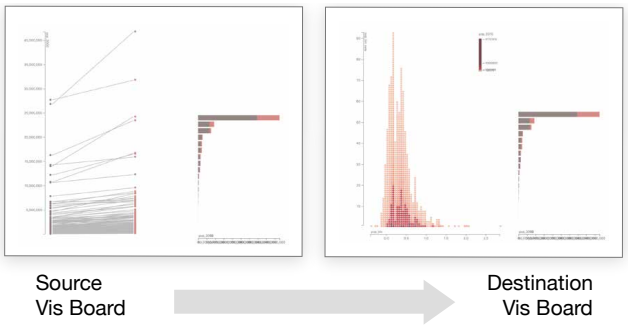


Figure 6: Enlarged view of the source and destination vis boards for the animation in Figure 2(e).

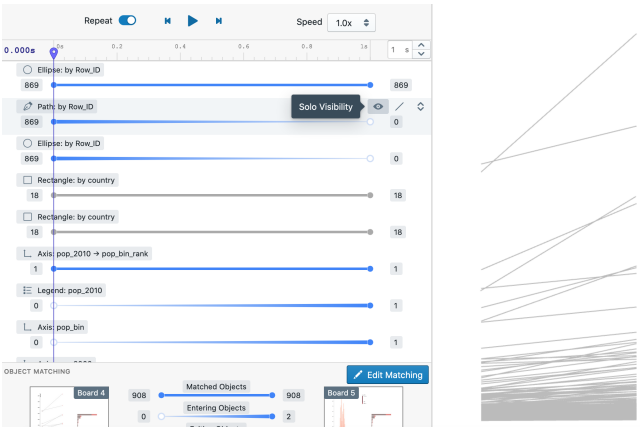


Figure 7: Hovering the mouse over the eye icon will show only the corresponding object set in the preview on the right.

superimposed over one another (each bar represents a country, the brown bars represent urban populations per country in 2000, and the red bars represent urban populations per country in 2010). In the destination vis board, the bar charts remain the same, but instead of a slope graph, we have a dot plot. The horizontal axis represents binned growth factor of population, and each dot represents a city. The color of the dots represents the city's population in 2010. The transition thus primarily happens between the slope graph and the dot plot (Figure 6).

In this example, there are many layers, each representing the matching result for one pair of object sets. Since it might be difficult for users to understand which object set each layer corresponds to, we added a feature to let users filter out object sets by hovering over the eye icon at the top right corner of each layer. For example, in Figure 7, the preview area on the right only shows the path object set in the slope graph. This path object set does not have a matching set in the destination vis board, because the slope graph is replaced by a dot plot. The timeline for the path object set thus has 0 thickness at the right end. This tapered representation shows that the path objects will be exiting in the destination vis board. Conversely, the legend for the dot plot (Legend:pop_2010, seventh layer in Figure 7) is absent in the source vis board, and it will be entering into the destination vis board. The timeline representation thus is tapered where the left end has no thickness.

4.3 Adjusting Matching Results through the Object Matcher

Since automated matching may fail in certain cases, or produce confusing transitions, Data Animator provides the Object Matcher view, a novel user interface for designers to interpret and adjust the matching results. Wrongly matched objects can be unmatched, and users can also manually create a match between two or more sets of objects.

Using the example in Figure 6 again, there are two sets of ellipses in the slope graph in the source vis board: the set on the left colored in gray represents cities in 2000, and the set on the right colored in

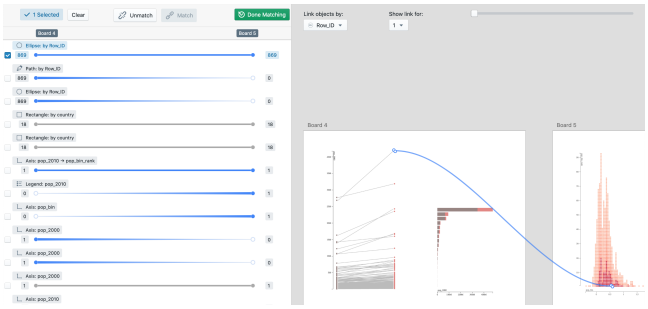


Figure 8: Selecting a matching result previews the links between objects.

red represents cities in 2010. In the destination vis board, there is only one set of ellipses in the dot plot. All these three sets have the same cardinality (869), and it would be a perfect match to map any of the two sets in the slope graph to the set in the dot plot. In this case, Data Animator creates a matching between the red ellipses in the slope graph and the ellipses in the dot plot (first timeline in Figure 7). This also leaves the gray anchor points without any match (the third layer in Figure 7). Logically this matching is correct, but the resulting animated transition looks odd as the red ellipses move through the center of the chart and cross over each other to get to their position in the dot plot. Designers may prefer instead having the slope graph fade out and the dot plot fade in, to reduce the number of flying objects on screen. Thus, the existing match needs to be removed.

To remove a match, users click the "Edit Matching" button in the lower part of the panel (Figure 7) to switch to the Object Matcher View (Figure 8), which is a modified version of the Timeline View. Users can select the matched object sets (in this case the first layer), and the preview area on the right shows how individual marks match across the source and destination vis boards using a blue link (Figure 8). After selecting the matched object sets, users click

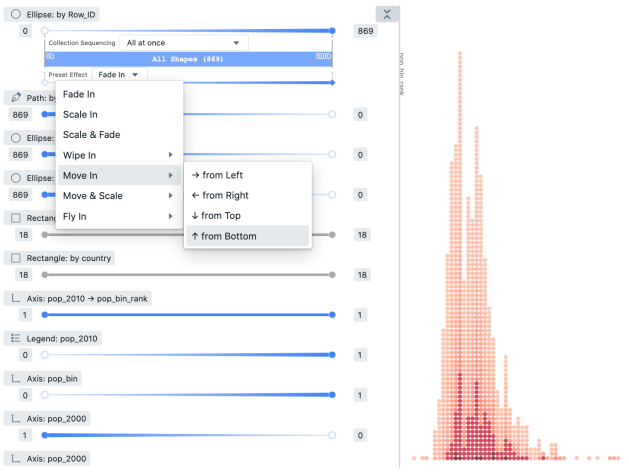


Figure 9: Interface controls for choosing an entering or exiting preset effect for an unmatched object set.

the "Unmatch" button at the top to break the matching. With the matching removed, users can go back to the Timeline View by clicking "Done Matching". They can control how the ellipses in the slope graph animate out (have the slope graph ellipses fall down with the "Move Out - to Bottom" preset) and then have the dot plot ellipses rise up with the "Move In - from Bottom" preset (Figure 9).

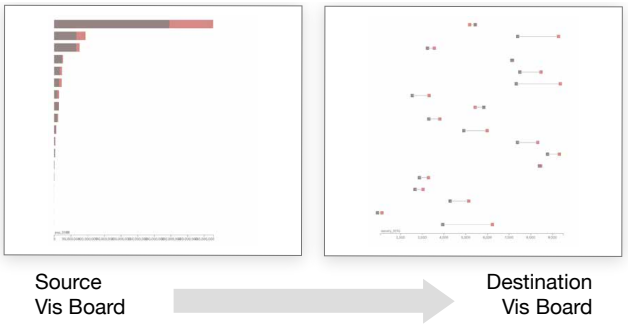


Figure 10: Enlarged view of the source and destination vis boards for the animation in Figure 2(g).

In another example where we need to transition from a superimposed bar chart to a dumbbell chart (Figure 10), the bar chart in the source vis board superimposes two bar charts, representing the population for each country in 2000 and 2010 respectively; the dumbbell chart shows the same information with different encodings: each country is a path, and the x position of the ellipses represents the population in 2000 and 2010 respectively. Data Animator does not make a match between any of the marks, because their types do not match at all. However, a match will be beneficial, as the animation maintains congruency so that the two sets of bars transition into the two sets of ellipses.

To manually create the matching, users click the "Edit Matching" button to switch to the Object Matcher View (Figure 11). Users select the two object sets they want to link, and then click the "Match" button. Data Animator will create a matching between the two sets and generate intermediate frames by tweening.

We refer the reader to the accompanying video that more fully illustrates the dynamics of these specifications and flow through the user interface.

Figure 11: Selecting two unmatched object sets to create a matching.

5 SPECIFYING TEMPORAL PACING OF ANIMATIONS

After matching the objects in the source vis board and the destination vis board, automatic animated transition through tweening

does not always achieve desired results. Animated data graphics often contain hundreds or more graphical objects, and it can be overwhelming for all these objects to start animating at once.

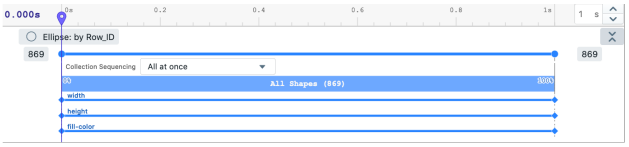
Data Animator supports the division of a complex animation into stages and uses data attributes to stagger the start time and vary the speed within sets of animating objects. The "Timeline Editor" in Data Animator differs from other timeline interfaces in design tools [2, 33] by allowing users to edit temporal properties of object sets, rather than tediously creating and adjusting each keyframe for tens or hundreds of objects. In this section we describe the temporal concepts of the system framework, and introduce a novel timeline interface for designers to pace the temporal rhythm of data-driven objects.

5.1 Staggering and Speeding by Data Attribute

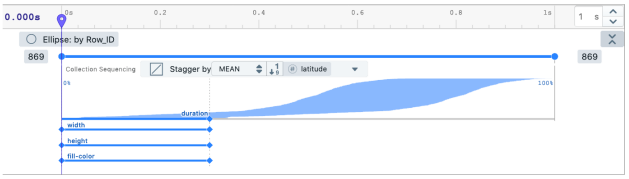
In a simple example like Figure 5, where the size and color of the ellipses in a scatter plot animate between two vis boards, designers may want to stagger the animation by introducing an incremental delay to the starting time of each ellipse. Instead of having to do so manually for all the 869 ellipses, they can use the "Expand" button in each timeline to show the user interface controls related to temporal pacing. By default, all the ellipses start at the same time with the same duration (Figure 12 (a)). Users can choose from the drop-down menu to either stagger the starting time or control the speed by an ordinal or quantitative data attribute. For example, users may want the cities at lower latitudes to animate first, so they can choose staggering by the attribute "latitude". The timeline visualizes how staggering will work by plotting a horizontal line for each ellipse, where the left and right ends of the line represents the starting and ending time of the animation. These lines form an overall shape depicting the distribution of starting and ending times of the ellipse set (Figure 12(b)). Alternatively, users may want to control the speed of animation by data, for example, by letting cities with greater population growth to animate more slowly. They can select "speed by pop_growth" from the drop-down menu, and the timeline shows a visual summary of the effect of this action (Figure 12(c)). When users stagger or speed by a quantitative attribute, they can also choose the aggregate (e.g., mean, max, min) if the object's data scope consists of more than one data row and has more than one value for the chosen attribute.

5.2 Staging

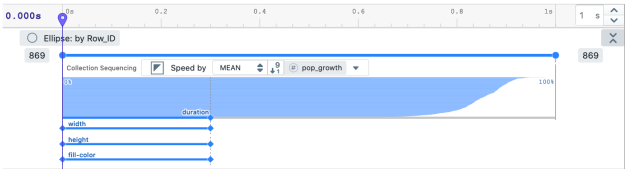
Staging is another commonly used technique to break down a complex animation into sub temporal groups. In the example in Figure 5, multiple visual properties are animating (width, height, and fill-color). Users may want to animate the size properties first before animating the fill-color. In Data Animator, each animating property has its own layer (Figure 12(a)). To create staging, users simply drag the end points of a timeline in a layer to change the starting and ending time of the animation for a property. In Figure 13 (a), two stages are created. In the first stage, the width and the height properties animate; in the second stage, the fill-color property animates. Data Animator also supports applying staggering after the stages are set up. In Figure 13 (b), we use the "latitude" data attribute to stagger the ellipses after the stages have been created in Figure 13 (a). Due to staggering, each ellipse will have less duration to animate, and the resulting durations for the two stages are allocated proportionately according to the original staging design. Figure 13 (b) shows how this information is conveyed in the timeline view.



(a) All objects start animating at the same time with the same duration by default

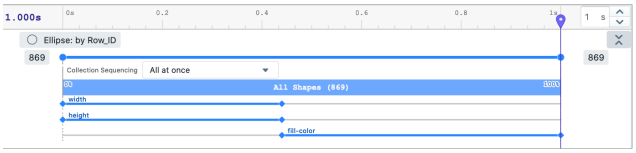


(b) Staggering objects by the data attribute "latitude"

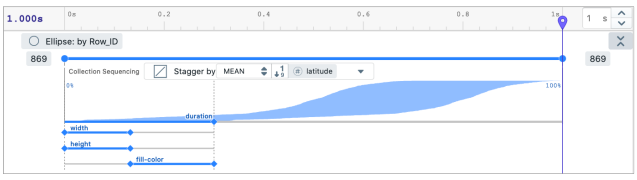


(c) Set the speed of animation objects by the data attribute "pop_growth"

Figure 12: Interface controls for using data attribute values to modify the speed and staggering of animation effects. The middle modification makes the animation start times delay (stagger) as a function of the latitude data attribute. The bottom modification changes the speed to be a function of population growth.



(a) Two stages of animated transition: width and height first, fill-color next



(b) Staggered objects will keep the stages for the visual properties proportionately

Figure 13: Making modifications to create animation stages. In the top, the user creates a first stage of width and height change, followed by the fill-color changing. In the bottom view, stagger each animation as a function of data.

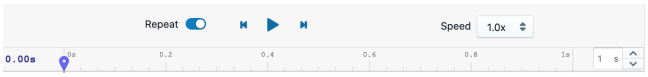


Figure 14: At the top of the left panel in the Timeline Editor, users specify the duration of animated transition between two vis boards

5.3 Hierarchical Keyframes

Figure 13(b) is an example of a novel concept we introduced in Data Animator: *hierarchical keyframes*, where the keyframes in a transition are constrained to the duration of its parent. The allotted time for each successive child cascades as a linear function of the parent's duration (a percentage value). There are three types of parent-child relationships in hierarchical keyframes: Object-Property, Transition-Object, and Object-Object.

- **Object-Property** - Objects with transitioning properties act as the parent timing for those differing property values. With percentage-based keyframes, the animation of properties can be staged one after another. Figure 13 (b) is an example of the Object-Property hierarchical keyframes, where the properties (as the children) inherit the allotted durations from the object (as the parent).
- **Transition-Object** - For a transition between two vis boards, multiple object sets may be involved. A global duration is specified as the root of the timing hierarchy. As shown in Figure 14, the default global duration is 1 second. When users change the duration, the axis updates accordingly. This global duration serves as the parent timing for all of the objects' transitions between the two vis boards, and all child objects' duration is expressed as a percentage of this global duration. To get rapid feedback on the current design, users can pause and play the animation preview, change the rendering speed (e.g., "0.5x" for half speed), and manually scrub the playback by clicking and dragging the playhead (Figure 14).
- **Object-Object** - Groups or collections from Data Illustrator are higher-level constructs that visually group and layout objects. They also serve as the parents of their members in hierarchical keyframes. For example, in Figure 15, the visualization has a group of two rectangles. The parent group object has a duration of 80% of the global duration. Each rectangle's duration accounts for 50% of their parent's duration, and each rectangle is staged one after another in the group. Changes to keyframes of the parent group will constrain the allotted time for both child rectangles while still preserving their staging. For an example of how the Object-Object relationship is used in an actual animated data graphics, please refer to the "Stephen Few's Box Plot" video in our gallery (http://data-animator.com/gallery/few_box_plot.html).

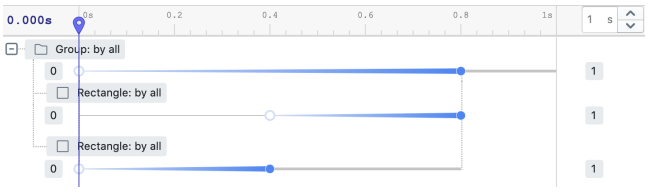


Figure 15: Parent objects such as groups constrain the allotted duration to child objects such as two rectangles.

5.4 Edit Temporal Pacing in the Timeline View: Scenarios

In this section, we present three authoring scenarios to illustrate how we can use hierarchical frames to create various staging and speeding effects.

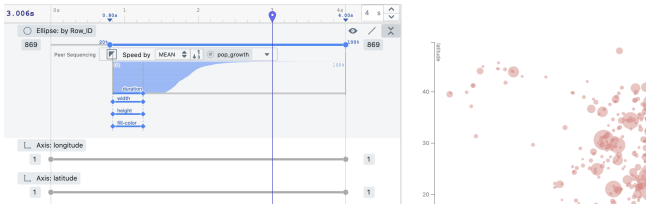


Figure 16: Pacing the animation for ellipses growing on a map. The user has delayed the start of the ellipses' movement and sets the speed to be a function of population growth.

Scenario 1: This first example is based on Figure 5, where the size of the ellipses animates to show data from `population_2000` to `population_2010`. The timeline consists of three layers: the matched ellipse sets, the matched x-axis for *longitude*, and the matched y-axis for *latitude*. Each layer consists of a source keyframe, a destination keyframe, and a path duration. For each layer, the object count is shown next to the source and destination keyframe (e.g., there are 869 *peer* ellipses matched from source to destination). Keyframes control the start and end times for all objects in the set to animate. In Figure 16, we click and drag the start keyframe to delay the set of ellipses from the start of the transition by 20%. The ellipses are now all delayed by 20%, simply dragging one keyframe sets the keyframes for all 869 ellipses in the set. In Data Animator, keyframes are assigned a percentage value rather in seconds. As mentioned in Section 5.3 this allows for relational timing to be preserved. When the user drags a keyframe the computed time in seconds appears on the timeline above. This feature is particularly useful for interpreting timing within hierarchies.

In this animation, by default, the ellipses animate uniformly within the set. To focus the viewers' attention on the ellipses that dramatically grow in size, we vary the speed of each animating ellipse based on the `population_growth` data attribute. Varying the speed creates an effect where the duration of each animating ellipse in the set is equivalent to how much the ellipse changes in size. To achieve this in the interface, we change the "Sequencing" from "All at once" to "Speed" by population growth rate. The sequencing updates to show the groupings created for population growth rate, and how they vary the speed (or duration). Playing the animation back, we notice it is too quick to perceive the changes in ellipse size. So we modify the duration of the entire transition to 4 seconds. All of the relative timings such as the delay and speed sequencing

are preserved due to the hierarchical keyframe approach of Data Animator.

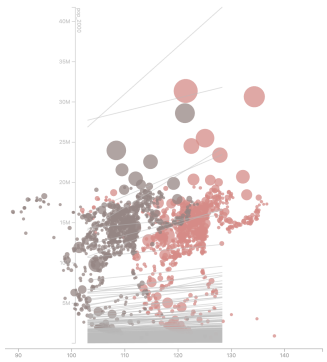


Figure 17: Transitioning from a symbol map to slope-chart without staging.

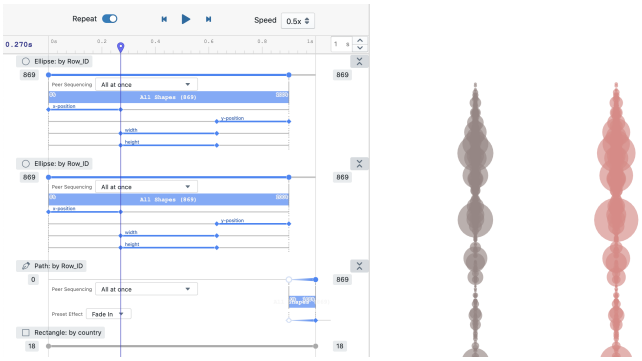


Figure 18: Coordinating multiple sets of unmatched objects with staging and staggering.

Scenario 2: Animating numerous sets of objects at the same time can result in a transition that is visually jarring and distracting to the viewer. For example, when transitioning from the symbol map to a slope-chart, the default timings results in Figure 17. The lines of the slope-chart intersect with the ellipses of the symbol map that match to the endpoints of the slope-chart. In the timeline we can coordinate the pacing of this animation to instead make small changes incrementally rather than one big transition. We start out by adding a delay to when the set of paths animate in to form the slope-chart, so we drag the start keyframe of this layer (third from the top in Figure 18) to 90%. Now that the paths our visually out of the way, we stage the matched ellipses before them. However, this staging does not solve our problem as the ellipses are animating too many visual properties at once. To stage these visual properties, we click to expand the properties of both sets of ellipses (top two in Figure 18). Expansion reveals the visual properties that differ between source and destination boards. Each animating property has its own pair of keyframes that we will use to stage this animation. We start by dragging the end keyframe for the x position to 30% in order to have it happen first. Next, we continue to stage width and height after x position and then create a final stage for the y-position. We redo the same staging process for the other set of ellipses in the timeline. And now this staging creates the effect of the symbol map clearing away from the middle of the canvas and stacking on top of each other as seen in Figure 18 (right). This staged transition is clearer than that in Figure 17. The timeline interface allows us to change the ordering of animating visual properties within sets of objects as another means to stage.

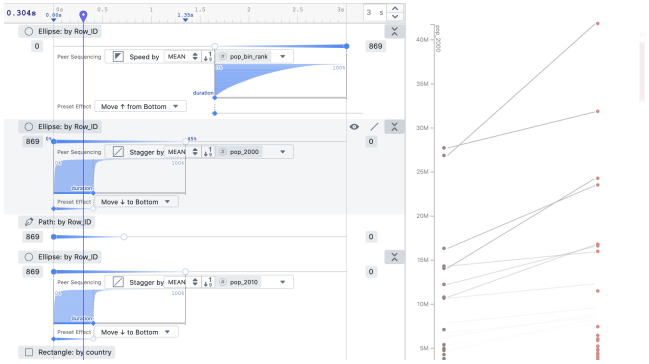


Figure 19: Coordinating multiple sets of unmatched objects with staging and staggering. The existing ellipses and paths of the slope-chart (lower timelines) animate first and are staggered by population. The new ellipses show up after that.

Scenario 3: The timeline editor also allows users to craft the pacing of unmatched objects entering and exiting from view. In this scenario, we use the example in Figure 6: two sets of ellipses and one set of paths that makeup a slope-chart are unmatched and exiting the scene. Additionally, one new set of ellipses enters into the scene to compose the dot plot. In Figure 19 we create staging between the two sets of exiting ellipses by adjusting both start keyframes to 45% and the start keyframe for the set of ellipses entering to be at 55%. Now the ellipses are staged before each other with a momentary pause of 10%. Instead of fading, we want the ellipses from the slope-chart to move down and out of the chart, and for the dot plot ellipses to rise up from the bottom – so we set both *preset effects* to be a move effect. Preset effects provide animation for unmatched layers that enter or exit. Data Animator supports fade, scale, scale & fade, move, move & scale, wipe, and fly preset animations. As is, the animation has better visual ordering, but all sets of ellipses occlude each other as they move. To combat this issue, we employ staggering for both sets of exiting ellipses based on population data attributes represented in the y axis. The ellipses from the dot plot have the same occlusion problem, but instead of using staggering we opt to sequence the speed of the dot plot ellipses related to their y position. With this sequencing the ellipses now animate at equivalent speed to the distances they have to travel. This gives the visual effect of the ellipses rising together – dropping off ellipses in their respective y positions as they move upward. Finally, we need to appropriately animate the set of lines that connects the slope-chart. Currently, the lines overlap with the entering dot plot, cluttering up the animation. We change the

end keyframe of this set of lines to be 10% to create the illusion that the lines hold the slope-chart together, and removing the lines releases the end point ellipses allowing them to fall. To time the lines with the falling end points we also sequence the lines based on the population in 2000. In this example we have gone from an undesirable transition created automatically to an animation that viewers can more easily follow with staging and sequencing of numerous sets of peer shapes.

6 IMPLEMENTATION

We implemented Data Animator as an HTML web application using a typical React [21] and redux [1] system architecture. The user interface (UI) components are bolstered by Blueprint.js [46], a React-based UI toolkit for data-dense web interfaces. With redux, the app maintains current (and previous states) for the animated data graphic specification edited by the user. The animated data graphic specification encapsulates all information needed to reproduce a design: vis boards (i.e., static Data Illustrator specifications), animated transition designs, and datasets.

The rendering engine takes the animated data graphic specification as input to render to an HTMLCanvas element with WebGL technologies. Our custom WebGL rendering engine is a re-usable component – used in the "Timeline Editor" and "Preview". The renderer component is based on the Three.js [14] and Three-BAS [65] libraries. Three-BAS (Buffer Animation System) is an extension of Three.js where shapes (e.g., rectangles, ellipses, paths) that share the same geometry but differ in visual properties (e.g., size, position, color) are rendered with animation logic in WebGL shaders. This paradigm improves frame-rate performance by storing visual properties of similar geometries, in our case peer shapes, on the Graphics Processing Unit (GPU). The rendered animation state for all peer shapes is then updated via custom vertex and fragment shaders on the GPU. This improves on typical WebGL applications that interpolate visual properties on the UI thread, creating a bottleneck that degrades frame-rate performance. In Data Animator, user changes to animating objects trigger an update to the properties stored in the GPU. However, the application state is separated from the animated data graphic specification to minimize updates to the rendering engine, only triggering updates to the React-based UI. We chose the Three-BAS architecture after evaluating the frame-rate performance of other JavaScript graphics libraries [4, 26, 28, 40, 50] against several expressive animated data graphic designs.

7 EVALUATION

7.1 Examples

To demonstrate the expressivity of our approach, we created a set of animated data graphics using the Data Animator system. The collection is available at: https://data-animator.com/gallery/index.html. Each example includes a completed project file, a video demonstrating the authoring process, and the final interactive version of the animated data graphic. The gallery consists of a diverse set of re-creations and variations from projects that sample the design space [57].

A gallery adequately showcases archetype designs, however, to show all possible designs would be time-consuming to create that number of examples. Instead, we comment on Data Animator's expressivity here with a brief description of the animated transition types that it supports. Data Animator currently supports 6 out of 7 transition types included in Robertson and Heer's taxonomy [31]. The current implementation supports transitions for substrate transformation, ordering, visualization changes, and data schema changes (orthogonal and nested). Data Animator also supports the specification of filtering (or highlighting) and timesteps by the user specifying multiple vis boards in Data Illustrator, each corresponding to a filtered state or time step. The system does not allow designers to generate these transitions from a single visualization template. In Section 8 we discuss future work to support the data-driven generation of multiple vis boards automatically by a data attribute. Data Animator does not support view transformations, lacking a user-directed camera to transition between viewpoints.

7.2 Usability Study

To evaluate the usability of Data Animator we conducted a re-creation study similar to the protocol used in related visualization authoring tools [42, 49]. In practice, designing animated data graphics involves gathering and cleaning a dataset, conducting data analysis to find insights, brainstorming the key visualizations and animations to convey those insights, and finally executing those ideas as an animated data graphic. A re-creation study focuses on that final execution step in the design process. We considered other study designs to evaluate Data Animator, one of those was an iterative, open-ended design task. For the purposes of evaluating a tool, re-creation tasks allow participants to experience authoring the same example animation and requires participants to test a larger range of the tool's features than they might otherwise. In an open-ended design task, there is no guarantee that participants will use the same breadth of features. In the case of our study, the tasks do not include designing static visualizations in Data Illustrator. Participants are provided with completed static visualizations to import into Data Animator for each task. While the re-creation task is not an exact replica of the design process, the ability to think and act in terms of our keyframe approach is the cornerstone of using Data Animator.

7.2.1 Participants. For the study, we recruited 8 participants with experience in graphic, animation, and visualization design. The participants (7 male, 1 female) reside in different geographic areas in the United States and their job titles range from UX designer, graphics designer, data journalist, grad student in HCI, and professor in data governance. The participants had varying years of experience in the related design disciplines:

- Graphic Design: none (0); less than 1 year (2); 1 to 2 years (1); 2 to 5 years (3); 5 to 8 years (1); more than 8 years (1)
- Animation Design: none (1); less than 1 year (2); 1 to 2 years (2); 2 to 5 years (3); more than 5 years (0)
- Visualization Design: none (0); less than 1 year (0); 1 to 2 years (3); 2 to 5 years (3); 5 to 8 years (1); more than 8 years (1)

Participants described the kinds of animated visualizations they create: interactive visualizations with animation (5); data stories or videos (3). To create animations, the participants most frequently use programming toolkits (4), Adobe After Effects [2] (3), and presentation tools such as Microsoft PowerPoint or Keynote (4). When

creating visualizations, the participants have employed a wide variety of tools such as vector editors (7); shelf builder interfaces like Tableau (6); spreadsheets such as Excel (6) or programming toolkits (5). When evaluating authoring tools with generative data support, it is important to have a mix of participants with programming and non-programming experience to see if programming knowledge affects participants' ability to understand concepts in Data Animator. Also, it is important to understand if prior experience with Data Illustrator would correlate to each participant's experience using Data Animator. The participants had varying degrees of experience with Data Illustrator: none (4); browsed the tutorials and tried once (1); used multiple times (3).

7.2.2 Procedure and Tasks. Each study lasted about 1.25 hours and was conducted remotely over video conferencing due to COVID-19. Because it was a remote study, participants used their own computers with varying performance metrics, screen sizes, and operating systems. Despite the difference in computing setups there were almost no usability issues due to any of these variables (one participant could not import through drag and drop). The only constant was that all participants used the Chrome web browser. During the session, participants shared their screens and were encouraged to think-aloud during each task. Audio and screen sharing feeds were recorded, but no user meta-data was logged by the system. We conducted two pilot studies to test our study protocol and all the study sessions were conducted by the first author.

Participants followed a 20-to-25-minute tutorial led by the moderator. The tutorial walked through a running example of four animations that transition through a series of bar charts, a bubble chart, and a stacked bar chart showing medal counts from the 2012 Olympic Games in London (http://data-animator.com/gallery/olympic_medals.html). Participants were prompted to take a break before moving on to the task portion of the study.

We then asked participants to re-create six non-trivial animations from two example data stories. Participants were provided videos of each animation task and the example visualization files to import into Data Animator. The first three tasks are based on Russell Goldenberg's "Twenty Years of the NBA Redrafted" data story [25] (study version: http://data-animator.com/gallery/nba_redraft.html). Tasks 1-3 require the participant to create three animations that transition through a series of four visualizations showing how the NBA draft from 1989 to 2008 could be redrafted based on past performance to reveal which players lived up to their draft hype or exceeded expectations. The four visualizations include two scatter-plots and two bubble charts. The second example takes inspiration from Tony Chu and Stephanie Yee's "A Visual Introduction to Machine Learning - Part 1" [64] (study version: http://data-animator.com/gallery/intro_to_ml.html). Tasks 4-6 ask participants to create three animations that reveal the beginnings of a decision-tree model that searches for the best data attribute to classify housing data. The visualizations include a scatter-plot, grouped uni-variate charts, a histogram, and a bar chart. Each task was presented to the participant as a video, animation plays back and includes a description on the side. Since we were not testing the participants' skills to deconstruct an animated visualization, we explained the dataset, data encodings, and animation pacing within each example. During pilot studies we found that a video of each animation without descriptions was insufficient for participants. Participants commented that it was difficult to decipher the pacing mechanisms of the animation. Therefore, we included text descriptions of the animation design within each video. At the end of the session, each participant completed a questionnaire to rate their experience (e.g., ease of use, learnability, enjoyment) and answered questions in a semi-structured interview about the moments they felt confused or empowered by Data Animator and how the tool could be improved or helpful for their current visualization practices.

The 6 re-creation tasks are meant to increase in difficulty. All participants completed each task with almost no help (except that P1 required assistance on Task 4). On the whole, participants completed the tasks quickly (μ=2 minutes 42 seconds, σ=1 minute 29 seconds). The time taken to complete each task is commensurate with the difficulty of the steps required as seen in Table 2.

7.2.3 Results. Participants rated their authoring experience with Data Animator on a 7-point Likert scale. Overall participants favored the usability of Data Animator: on authoring animations, μ=2.25, σ=1.16 (1-very easy, 7-very difficult); on overall experience, μ=1.63, σ=0.74 (1-very enjoyable, 7-very frustrating); on learning, μ=3, σ=1.69 (1-very easy, 7-very difficult). As expected, the participants rated the re-creation tasks to be complex, μ=6.63, σ=0.52 (1-very simple, 7-very complex).

All participants found Data Animator to be useful for creating animated data graphics. The participants praised how Data Animator "*treats data as a first-class citizen*" (P7) as it leverages the underlying data to automatically match objects between visualizations and design pacing effects based on data attributes. In particular, participants noted how Data Animator fills the gap between programming an animated data graphic and using animation design tools: "*The speed and flexibility at which you can [create animations] with Data Animator compared to writing custom code or having to wrestle with a design tool that is not meant for working with data is amazing*" (P5). The participants with programming knowledge commented how Data Animator provides comparable results to coding in a more streamlined approach "*If I wanted to create something that looked like this [with code] would be quite a headache. Being able to change what you see and play with the animations is pretty great*" (P6). They also compared the expressiveness of Data Animator to related visualization specific tools: "*Tableau's animations are cool and magical when they get it right, but you have very little avenues for when they get the animation wrong. So, I appreciate that Data Animator takes that into account*" (P7). Participants felt empowered to be able to create animations without having to program: "*As someone who doesn't code, I have to dream about creating [static] charts because there are so few tools that I can use. And then animating charts is like a dream within that dream! [Animating] is something that is completely outside of what I can expect, unless I'm willing to animate by hand in After Effects. So yes, [Data Animator] is tremendously useful.*"

The timeline interface felt natural and familiar to participants with experience using similar keyframing interfaces. The participants appreciated that Data Animator includes all of the "entry level stuff" for creating keyframe animations such as easings and

Table 2: Task completion results for Tasks 1-6 of usability study.

	Completion Times (minutes:seconds)			Data Animator Features Used Per View		
Task	00:00 02:00 04:00	Mean (μ)	SD (σ)	Storyboard	Timeline Editor	Object Matcher
1		00:54	00:28	create transition	stagger shapes	
2		02:17	00:53	create transition	stagger shapes; stage properties	
3		02:53	00:57	create transition	stage properties; stage axes	
4		03:27	01:35	create transition	stagger shapes; stage properties; stage axes	match shapes
5		03:44	01:40	create transition	stage shapes/collection; stagger shapes; stagger collections; set preset effect; stage axes	
6		02:55	01:18	create transition	stagger shapes; stage properties; stage axes/legends	

adjusting keyframes - but also provides novel user interfaces to create staggering and speeding by data that they "*have not seen before*" (P2). All participants commented on the ease at which they could coordinate the timing of sets of objects all by changing one keyframing. Many of them described how the same process would be "time-consuming", "brutal" and "painful" to do by hand in design tools such as Adobe After Effects.

The participants also suggested additional features to perfect the timeline editing experience. Many asked for the ability to snap keyframes to nearby keyframes or adjust multiple keyframes together with selection. P5 and P7 also proposed the idea of a stage component that could divide the overall transition into sub-transitions which could alleviate the need to align keyframes when staging.

All participants were able to comprehend the results of the automatic matching algorithm. Many participants felt that this matching feature was similar to other auto-animate or "Magic Move" features from related tools [3, 6] For example, P2 commented on how morphing the ellipses to rectangles in Task 4 would be "*super hard and there is no fix [in other tools] - you have to do it frame by frame in After Effects*", and they appreciated the idea of Data Animator to use data to make that matching for you. Despite understanding and appreciating the results of the automatic matching algorithm, 5 participants found the interface to manually change matches to be confusing. P7 commented that "*all the concepts are there, but it lacks the affordances in the user interface to understand what you should do*". Also, P2 commented that "*it would be nice to see what happens after you make a match – I'm guessing that was the right thing to do, but the interface doesn't immediately clarify that for me*". Future improvements to Data Animator should consider improving the "Object Matcher" interface by: positioning interface elements as to guide the user in the expected order of operations, create affordances surrounding the layers to select, clarify which action button to click next, and improve feedback when matches are created or removed.

When asked about how they would use Data Animator, the majority of participants responded that they would use it for presentation of data. P7 explains: "*my favorite visualizations takes something very complex and breaks it down piece-by-piece. So, something like VORP from the NBA example, nobody gets what that is but if you take some very well-known metrics and walk them step by step to how they relate... they will at least be closer to understanding that complex concept... [I create] Keynote decks where a concept builds on top of another concept [and so on]. I usually don't use real data for [those animations] but would I ever if I could!*" Participants commented on additional uses outside of presenting data. P6 felt that it could be helpful for teaching data analysis – allowing students to create more expressive representations of their data. They also explained how a tool like Data Animator could help improve data literacy – as a broader user group (designers) would be able to create new types of visualizations. Even participants with programming knowledge felt that Data Animator would save "a lot of time" to prototype animations that they would eventually program in a fully interactive web page (P5).

8 DISCUSSION AND FUTURE WORK

A major prerequisite of using Data Animator is that the static visualizations need to be in the Data Illustrator format. We acknowledge the vital role that Data Illustrator provides in our approach: producing static visualization keyframes. Data Animator's role is to declare animated transitions between two static visualizations, which is the basis for the keyframe animation approach. This assumption can be limiting, as Data Illustrator is not necessarily integrated in many designers' workflows. However, basic vector graphics formats such as SVG are not sufficient to author animated data graphics. It is essential to provide information about data binding, visual object grouping and hierarchy so that the system can automate the matching of objects between static graphics. To date, there has been no universal format that describes complete information about a static data visualization, and different authoring systems for animated data graphics are adopting different standards or formats. For example, Canis [24] requires a data-enriched variant of SVG to be used as input format. We chose the Data Illustrator framework because it also includes information about object grouping and hierarchy in addition to data binding, and such information is useful when designers author temporal pacing that depends on the structure of visualizations. In future work, we are interested in developing tools that translate visualizations in other formats to the Data Illustrator format. Such a tool may parse a vector graphic, analyze the properties and structure of visual objects, and ask designers to provide minimal annotations. The output of such a tool can be directly fed into Data Animator as input static visualizations.

The dependency on the Data Illustrator file format does not mean that users of Data Animator need to know how to use Data Illustrator. 5 of the 8 participants in our user study have minimal experience in using Data Illustrator, but all of them could complete the tasks in a few minutes without help. We have thus found no

qualitative evidence that prior experience with Data Illustrator affects the use of Data Animator. This finding suggests that Data Animator has the potential to reach a broad set of users who are not necessarily familiar with Data Illustrator.

Users desire the capability to share their animated data graphics with others via the web. We intend to support the export of animated data graphics created in Data Animator as template web pages. At the time of publication, this feature had not been fully realized and is earmarked for future system updates. The export process will allow authors to produce an interactive web page of their data narrative, similar to the "Preview" panel of the system. Authors will support navigation between vis boards in their data story via user input (e.g., button-clicks, scrolling). The exported HTML page will contain a single canvas element to display the animated data graphic and user input elements. We plan to include a custom JavaScript plugin library to handle generating, rendering, and navigating an exported animated data graphic specification. The plugin library is a pared-down version of the rendering engine and playback features supported by Data Animator.

The main challenges to implementing export capabilities are file size and rendering performance. Exported animated data graphics comprise multiple static visualization keyframes, each containing meta-data for the graphical marks, hierarchies, data tables, data scopes, and visual encodings. Compression techniques should be used to remove redundancies from these keyframes and to ensure manageable file sizes for hosting web-based content. Second, the rendering capabilities of the exported web page should render at an adequate frame rate (45-60 frames per second). The current Data Animator rendering engine based on three.js [14] and Three-BAS [65] achieves this level of performance. We plan to re-package this rendering code as part of the export plug-in.

Our primary focus in this paper is on authoring transitions between two static graphics under the keyframing paradigm. More work is needed to further enhance the expressivity of Data Animator. For example, keyframes in animations like a bar chart race are generated by updating a bar chart template with a temporal attribute. Using Data Animator, creating a bar chart race would be very tedious, because designers need to prepare multiple vis boards, each corresponding to a different visualization state. Designers need to be able to import a visualization template and generate multiple vis boards automatically by a data attribute. A procedural paradigm would be appropriate for such functionality, as in the case of *transition_time* and *transition_states* in gganimate [41]. In addition, animated data graphics can benefit from techniques such as adding highlights and annotations, dynamically filtering objects by data, and syncing animated transitions with text explanations and voice narration. We plan to add these features to Data Animator in the future.

9 CONCLUSION

We present Data Animator, an authoring system for creating animated data graphics. The design of Data Animator focuses on two challenges: 1) establish matching between graphical objects on two static visualizations as key frames so that tweening, entering or exiting effects can be applied automatically, and 2) enhance the expressive power of the animated transitions by supporting staggering and staging that are driven by data attributes or data hierarchies. A study with 8 designers shows that Data Animator is learnable and usable: all the participants can complete each of the six non-trivial animation authoring tasks in a few minutes with minimal help. Feedback from the participants suggest further enhancement is required to improve the Object Matcher interface.

REFERENCES

[1] Dan Abramov. 2020. Redux – A Predictable State Container for JS Apps. http://redux.js.org/

[2] Adobe Systems Inc. 2020. Adobe After Effects. http://www.adobe.com/products/aftereffects.html

[3] Adobe Systems Inc. 2020. Adobe XD. http://www.adobe.com/products/xd.html

[4] Airbnb Inc. 2020. Lottie. http://airbnb.design/lottie/

[5] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Andres Monroy-Hernandez, and Pourang Irani. 2017. Authoring Data-Driven Videos with DataClips. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 501–510. https://doi.org/10.1109/TVCG.2016.2598647

[6] Apple Inc. 2020. Keynote - Apple. http://www.apple.com/keynote/

[7] Daniel Archambault, Helen Purchase, and Bruno Pinaud. 2010. Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs. *IEEE Transactions on Visualization and Computer Graphics* 17, 4 (2010), 539–552. https://doi.org/10.1109/TVCG.2010.78

[8] Bohemian Coding. 2020. Sketch - The Digital Design Toolkit. https://www.sketchapp.com/

[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[10] Matthew Brehmer, Bongshin Lee, Benjamin Bach, Nathalie Henry Riche, and Tamara Munzner. 2017. Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE Transactions on Visualization and Computer Graphics* 23, 9 (2017), 2151–2164. https://doi.org/10.1109/TVCG.2016.2614803

[11] Matthew Brehmer, Bongshin Lee, Nathalie Henry Riche, David Tittsworth, Kate Lytvynets, Darren Edge, and Christopher White. 2019. Timeline Storyteller: The Design & Deployment of an Interactive Authoring Tool for Expressive Timeline Narratives. In *In proceedings of the Computation + Journalism Symposium*. C + J, Boston, MA, USA, 1–5. https://aka.ms/TSCJ19

[12] Matthew Brehmer, Bongshin Lee, Petra Isenberg, and Eun Kyoung Choe. 2019. A Comparative Evaluation of Animation and Small Multiples for Trend Visualization on Mobile Phones. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 364–374. https://doi.org/10.1109/TVCG.2019.2934397

[13] Nadieh Bremer and Marlieke Ranzijn. 2015. Urbanization in East Asia between 2000 and 2010. http://nbremer.github.io/urbanization/

[14] Ricardo Cabello. 2020. three.js – JavaScript 3D library. http://threejs.org/

[15] Fanny Chevalier, Pierre Dragicevic, and Steven Franconeri. 2014. The Not-so-Staggering Effect of Staggered Animated Transitions on Visual Tracking. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2241–2250. https://doi.org/10.1109/TVCG.2014.2346424

[16] Tarik Crnovrsanin, Shilpika, Senthil Chandrasegaran, and Kwan-Liu Ma. 2020. *Staged Animation Strategies for Online Dynamic Networks*. IEEE. https://doi.org/10.1109/TVCG.2020.3030385 To appear in IEEE Transactions on Visualization and Computer Graphics.

[17] Datawrapper GmbH. 2020. Create charts and maps with Datawrapper. https://www.datawrapper.de/

[18] Density Design and Calibro. 2020. Raw Graphs. http://rawgraphs.io/.

[19] Pierre Dragicevic, Anastasia Bezerianos, Waqas Javed, Niklas Elmqvist, and Jean-Daniel Fekete. 2011. Temporal Distortion for Animated Transitions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. ACM, New York, NY, USA, 2009–2018. https://doi.org/10.1145/1978942.1979233

[20] Fan Du, Nan Cao, Jian Zhao, and Yu-Ru Lin. 2015. Trajectory Bundling for Animated Transitions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '14)*. ACM, New York, NY, USA, 289–298. https://doi.org/10.1145/2702123.2702476

[21] Facebook Inc. 2020. React – A JavaScript library for building user interfaces. http://reactjs.org/

[22] Figma Design. 2020. Figma: the collaborative interface design tool. https://www.figma.com/

[23] Danyel Fisher. 2010. Animation for Visualization: Opportunities and Drawbacks. In *Beautiful Visualization Edition*. O'Reilly Media, Sebastopol, CA, USA. https://www.microsoft.com/en-us/research/publication/animation-for-visualization-opportunities-and-drawbacks/

[24] Tong Ge, Yue Zhao, Bongshin Lee, Donghao Ren, Baoquan Chen, and Yunhai Wang. 2020. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum* 39, 3 (2020), 607–617. https://doi.org/10.1111/cgf.14005

[25] Russell Goldenberg. 2017. Twenty Years of the NBA Redrafted. http://pudding.cool/2017/03/redraft/

[26] Google Data Arts team. 2020. Two.js. http://two.js.org/

[27] Google News Lab. 2020. Visualizing Google Data. http://trends.google.com/trends/story/US_cu_6fXtAFIBAABWdM_en

[28] Mat Groves. 2020. PixiJS. http://www.pixijs.com/

[29] David Guilmaine, Christophe Viau, and Michael J McGuffin. 2012. Hierarchically Animated Transitions in Visualizations of Tree Structures. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12)*. ACM, New York, NY, USA, 514–521. https://doi.org/10.1145/2254556.2254653

[30] Jonathan Harper and Maneesh Agrawala. 2018. Converting Basic D3 Charts into Reusable Style Templates. *IEEE Transactions on Visualization and Computer Graphics* 24, 3 (2018), 1274–1286. https://doi.org/10.1109/TVCG.2017.2659744

[31] Jeffrey Heer and George Robertson. 2007. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1240–1247. https://doi.org/10.1109/TVCG.2007.70539

[32] Edwin Hutchins, James Hollan, and Donald Norman. 1985. Direct Manipulation Interfaces. *Human–Computer Interaction* 1, 4 (1985), 311–338. https://doi.org/10.1207/s15327051hci0104_2

[33] InvisionApp Inc. 2020. InVision. http://www.invisionapp.com/

[34] Adam Katz. 2020. Layer Repeater - aescripts. https://aescripts.com/layer-repeater/

[35] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. ACM, New York, NY, USA, 395–405. https://doi.org/10.1145/2642918.2647375

[36] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: Bringing Life to Illustrations with Kinetic Textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 351–360. https://doi.org/10.1145/2556288.2556987

[37] Kiln Enterprises Ltd. 2020. Flourish Studio. http://flourish.studio

[38] Younghoon Kim, Michael Correll, and Jeffrey Heer. 2019. Designing Animated Transitions to Convey Aggregate Operations. *Computer Graphics Forum* 38, 3 (2019), 541–551. https://doi.org/10.1111/cgf.13709

[39] Younghoon Kim and Jeffrey Heer. 2020. Gemini: A Grammar and Recommender System for Animated Transitions in Statistical Graphics. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2020), 1–10. https://doi.org/10.1109/TVCG.2020.3030360

[40] Jürg Lehni and Jonathan Puckey. 2020. Paper.js. http://paperjs.org/

[41] Thomas Lin Pedersen and David Robinson. 2020. A Grammar of Animated Graphics: gganimate. http://gganimate.com/

[42] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173697

[43] Microsoft Corp. 2020. Microsoft Powerpoint. http://products.office.com/en-us/powerpoint

[44] MW Motion. 2020. Blend - aescripts. https://aescripts.com/blend/

[45] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, Philadelphia, PA, USA.

[46] Palantir. 2020. Blueprint – A React-based UI toolkit for the web. http://blueprintjs.com/

[47] Casey Reas and Ben Fry. 2006. Processing: Programming for the Media Arts. *AI & Society* 20, 4 (Aug 2006), 526–538. https://doi.org/10.1007/s00146-006-0050-9

[48] Donghao Ren, Tobias Hollerer, and Xiaoru Yuan. 2014. iVisDesigner: Expressive Interactive Design of Information Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (December 2014), 2092–2101. https://doi.org/10.1109/TVCG.2014.2346291

[49] Donghao Ren, Bongshin Lee, and Matthew Brehmer. 2019. Charticulator: Interactive Construction of Bespoke Chart Layouts. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 789–799. https://doi.org/10.1109/TVCG.2018.2865158

[50] Donghao Ren, Bongshin Lee, and Tobias Höllerer. 2017. Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering. *Computer Graphics Forum* 36, 3 (2017), 179–188. https://doi.org/10.1111/cgf.13178

[51] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. 2008. Effectiveness of Animation in Trend Visualization. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1325–1332. https://doi.org/10.1109/TVCG.2008.125

[52] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum* 33, 3 (2014), 351–360. https://doi.org/10.1111/cgf.12391

[53] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2020. Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 461–471. https://doi.org/10.1109/TVCG.2019.2934281

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

[55] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (January 2016), 659–668. https://doi.org/10.1109/TVCG.2015.2467091

[56] Charles Stolper, Minsuk Kahng, Zhiyuan Lin, Florian Foerster, Aakash Goel, John Stasko, and Duen Horng Chau. 2014. GLO-STIX: Graph-Level Operations for Specifying Techniques and Interactive eXploration. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2320–2328. https://doi.org/10.1109/TVCG.2014.2346444

[57] John Thompson, Zhicheng Liu, Wilmot Li, and John Stasko. 2020. Understanding the Design Space and Authoring Paradigms for Animated Data Graphics. *Computer Graphics Forum* 39, 3 (2020), 207–218. https://doi.org/10.1111/cgf.13974

[58] Tumult. 2020. Tumult Hype 4.0. http://www.tumult.com/hype/

[59] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. 2002. Animation: can it facilitate? *International Journal of Human-Computer Studies* 57 (2002), 247–262. https://doi.org/10.1006/ijhc.1017

[60] Bret Victor. 2013. Drawing Dynamic Visualizations. https://vimeo.com/66085662

[61] Yong Wang, Daniel Archambault, Carlos Scheidegger, and Huamin Qu. 2018. A Vector Field Design Approach to Animated Transitions. *IEEE Transactions on Visualization and Computer Graphics* 24, 9 (2018), 2487–2500. https://doi.org/10.1109/TVCG.2017.2750689

[62] Hadley Wickham. 2009. *ggplot2: Elegant Graphics for Data Analysis*. Springer Nature, Houston, TX, USA.

[63] Hadley Wickham. 2010. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28. https://doi.org/10.1198/jcgs.2009.07098

[64] Stephanie Yee and Tony Chu. 2015. A visual introduction to machine learning. http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

[65] Szenia Zadvornykh. 2020. three.bas – THREE Buffer Animation System. http://github.com/zadvorsky/three.bas