CAST: Authoring Data-Driven Chart Animations

Tong Ge
Shandong University
Qingdao, China
tgeconf@gmail.com

Bongshin Lee
Microsoft Research
Redmond, United States
bongshin@microsoft.com

Yunhai Wang
Shandong University
Qingdao, China
cloudseawang@gmail.com

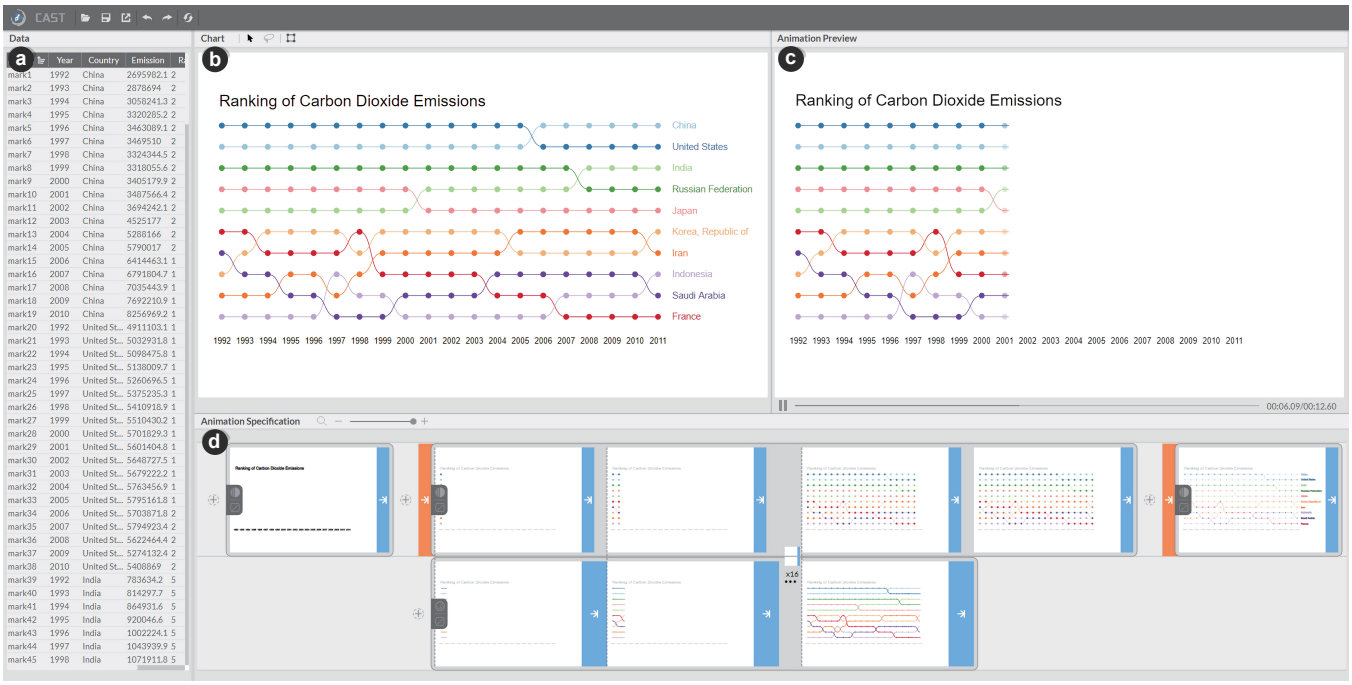


Figure 1: CAST enables the interactive construction of a variety of chart animations. Its interface consists of four panels: (a) data panel; (b) chart panel; (c) animation preview panel; and (d) animation specification panel. In this case, the system is about halfway through the animation of a *'bump chart,'* showing how the ranking of carbon dioxide emissions has changed over the years. Please visit the CAST website (https://chartanimation.github.io/cast) to see the animation.

ABSTRACT

We present CAST, an authoring tool that enables the interactive creation of chart animations. It introduces the visual specification of chart animations consisting of keyframes that can be played sequentially or simultaneously, and animation parameters (e.g., duration, delay). Building on Canis [19], a declarative chart animation grammar that leverages data-enriched SVG charts, CAST supports auto-completion for constructing both keyframes and keyframe sequences. It also enables users to refine the animation specification (e.g., aligning keyframes across tracks to play them together, adjusting delay) with direct manipulation and other parameters for animation effects (e.g., animation type, easing function) using a control panel. In addition to describing how CAST infers recommendations for auto-completion, we present a gallery of examples to demonstrate the expressiveness of CAST and a user study to verify its learnability and usability. Finally, we discuss the limitations and potentials of CAST as well as directions for future research.

CCS CONCEPTS

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

KEYWORDS

Chart animation, chart animation authoring, chart animation specification, data visualization, interactive system

ACM Reference Format:
Tong Ge, Bongshin Lee, and Yunhai Wang. 2021. CAST: Authoring Data-Driven Chart Animations. In *CHI Conference on Human Factors in Computing*

CHI '21, May 08–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445452

Systems (CHI '21), May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3411764.3445452

1 INTRODUCTION

Chart animations are an engaging means for communicating data-driven insights as they are effective in attracting and retaining audiences' attention. Hans Rosling's animated bubble charts [45, 46] and numerous compelling examples [3, 56] from practitioners gained huge popularity from the public, demonstrating a competitive advantage over static charts in increasing audience engagement.

Today, several commercial and research tools allow people without programming skills to create chart animations. However, most interactive tools for authoring chart animations (e.g., DataClips [4], Flourish [28], Adobe Stock [2]) ask users to choose from a set of predefined templates. They cover only a small number of standard chart types, such as bar charts, line charts, or pie charts, and provide limited support for customization in sequencing keyframes and pacing between them. As a result, these tools preclude expressive chart animations, preventing users from leveraging the wide range of charts that can be created with bespoke chart creation tools. Although general animation creation tools (e.g., Adobe After Effects [1]) can be used to author expressive chart animations, they often require tedious and time-consuming manipulation due to the lack of data-driven abstractions.

Besides interactive tools, people can use programming libraries like D3 [9] and gganimate [63] to author highly sophisticated chart animations. This approach, however, is only accessible to people with advanced programming skills and requires significant efforts in fine-tuning the animation factors, such as transition and pacing. To address this issue, Ge et al. recently introduced Canis [19], the first high-level language for the declarative specification of chart animations. In contrast to D3, Canis employs a simpler syntax for creating expressive animations by focusing on chart animations and leveraging data-enriched scalable vector graphics (SVG) charts as the input. However, it still requires people to write code to define keyframes and handle some of the timing factors.

In this paper, we present CAST (**Ca**nis **St**udio; Figure 1), a web-based authoring tool that enables people to create chart animations with a wide range of chart designs without programming. Building on Canis, CAST works with the charts created by existing chart construction tools. To facilitate the authoring and understanding of chart animations, it introduces a *visual specification* approach that explicitly represents keyframe-based chart animations with four visual components: keyframe, keyframe group, timing, and effect. Combining with a sequence-based timeline and storyboard, the specifications consist of keyframes that can be played sequentially or simultaneously, and animation parameters (e.g., duration, delay). To support the easy construction of keyframe-based animations, CAST offers data-driven *auto-completion* that suggests candidates for both keyframes and keyframe sequences. (We explain how the system infers candidates in Section 3.4 in detail.) Specifically, CAST takes data-enriched SVG charts as the input and enables authors to craft animations using three key features: keyframe construction, keyframe sequencing, and animation property refinement. After constructing keyframes and sequences with the auto-completion through a few selection operations, authors can refine the animation specification (e.g., aligning keyframes across tracks to play them together, adjusting gaps between frames) with direct manipulation. Other parameters for animation effects (e.g., animation type, easing function) can be adjusted using a control panel. The easy-to-understand visual specification of chart animations and this simple form of user interaction powered by the auto-completion make CAST accessible to novices who lack the ability to program chart animations.

In summary, the main contributions of this work are:

- We introduce the visual specification of chart animations that explicitly represents keyframe-based chart animations. It conveys the core aspects of animation with four components—keyframe & keyframe group, timing, and effect.
- Based on our visual specification, we design and develop CAST, a web-based system that enables the interactive construction of chart animations. CAST employs data-driven auto-completion to reduce the effort required in constructing keyframes and keyframe sequences, and direct manipulation to facilitate the easy specification of chart animations.
- We present two forms of evaluations. Our gallery (Figure 2) demonstrates the expressiveness of CAST. Our two-part user study with 18 participants assesses if participants could learn and understand our visual specification, and evaluates the learnability and usability of CAST.

2 RELATED WORK

CAST builds on visualization system research spanning perceptual guidelines, tools for authoring chart animations, visual programming language, and the underlying chart animation language, Canis [19]. In this section, we focus on the creation of chart animations and the reader can refer to previous studies [6, 10, 43] for the effectiveness of chart animations.

2.1 Perception of Animations

A complete review of general perception of animations is beyond the scope of this paper (refer to Tversky et al. [57] and Chevalier et al. [13]). We restrict our discussion to the perception of data-driven chart animations, where motion is used as a preattentive visualization technique [59]. Previous studies show that appropriately-designed animations not only reveal complex data relations (e.g., causal relations [60]) but also support filtering and brushing [7] as well as facilitate decision making [20].

Following the Gestalt law of common fate [26], the animated visual marks moving with the same velocity are perceived as the same group. A few studies investigated the human abilities in perceiving the changes of speed [31], direction [55], or both [22] in animated visualization. Weiskopf [61] found that color plays an important role in the perception of motion direction and velocity. Romat et al. [44] introduced three motion variables (speed, frequency, and pattern) for defining animated edge textures in node-link diagrams and assessed the effectiveness of each variable. Recently, Chalbi [11] suggested that not only the motion but also the change in luminance and size all have the strong grouping effect. Although CAST does not fully guarantee the perceptual effectiveness of chart animations,

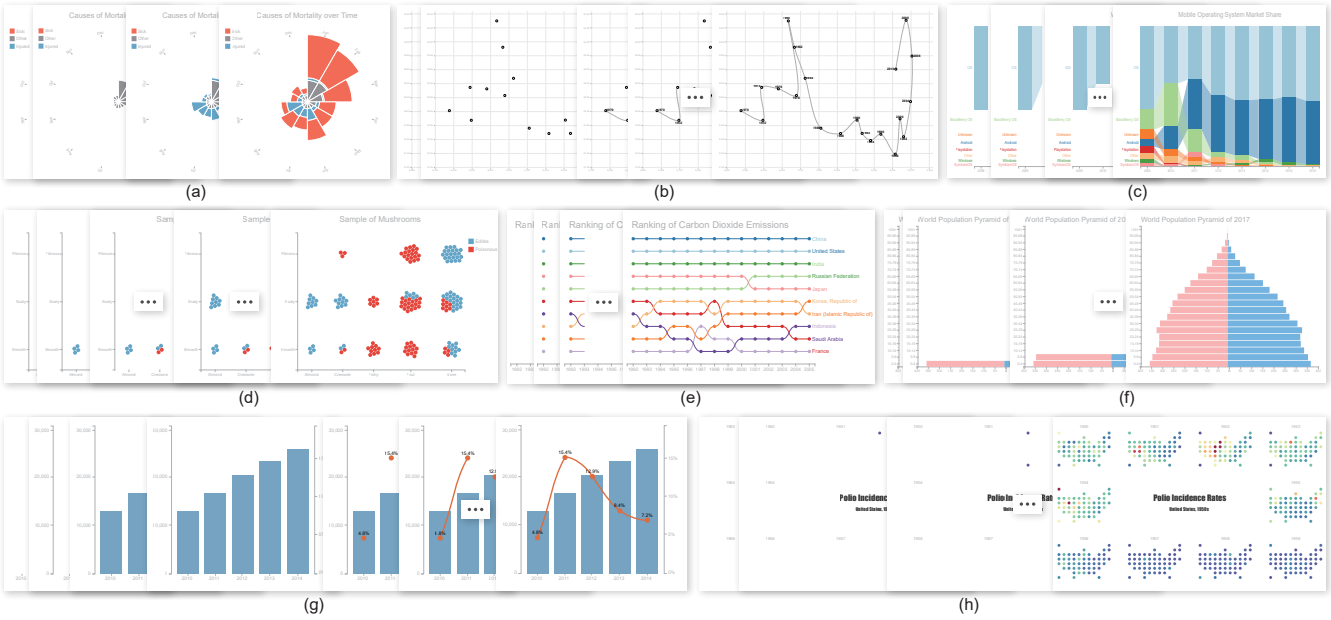


Figure 2: Eight example chart animations demonstrating the expressiveness of CAST. We used the first seven chart animations (a-g) as tasks in our user study in Section 4. In addition to these, more examples can be found with the input charts, descriptions, and videos illustrating the creation processes in the website (https://chartanimation.github.io/cast).

we were mindful about the previous research on animation perception: CAST's auto-completion ensures the perceptual grouping by analyzing the visually encoded data attributes.

2.2 Chart Animation Design Space

Animation is a promising way for conveying the changes in visualizations, although its effectiveness is still controversial for data analysis [57]. To promote effective animation design, a few guidelines have been proposed. Tversky et al. [57] suggest two high-level principles: congruence and apprehension, namely, requiring that the animation should be accurately perceived with respect to user's mental representation. While adhering to these principles, Heer and Robertson [23] and Fisher [18] recommend a few specific design guidelines for crafting effective chart animations including "consistent semantic-syntactic mappings" and "meaningful motion." Through carefully examining a corpus of data videos, Amini et al. [3] characterize the elemental units of data videos as the combinations of visualization types × animation types. Very recently, Thompson et al. [56] characterize the design space by four dimensions: object, graphic, data, and timing. The object dimension refers to which marks undergo an animation, the graphic and data dimensions describe the change of visual states of the marks, and the time dimension specifies the pace sequences of the animation. Those dimensions can be combined into compositions of animated transitions and pacing techniques. While not perfectly aligned with Thompson et al.'s design space,[1] CAST's design can be projected onto their space. CAST enables users to select *graphic objects*, such as marks, (title) text, axis, legend, to be animated, and use them to construct keyframes and keyframe sequences by *data-driven* auto-completion. In addition, users can specify the animation effects (e.g., appear) and modify the *timing* of animation (e.g., duration, delay).

The effectiveness of different aspects of animation transition design, such as staging, staggering, and trajectories, have also been studied. Bartram and Ware [7] find that objects with similar motions are preattentively grouped even they are otherwise dissimilar. Heer and Robertson [23] show that carefully designed staged animation improves understanding of the underlying data, while complex multi-stage transitions are less favored. Shanmugasundaram et al. [50] find that smooth animation transitions substantially help in maintaining the connectivity and overall structure in node-link diagrams. Dragicevic et al. [14] compare the effects of different temporal distortion strategies on object tracking and suggest that slow-in/slow-out outperforms others, although the differences depend on the type of animation transitions. Chevalier et al. [12] study another pacing technique, staggering, where the start time of moving elements is delayed incrementally and found that it has a negligible, or even negative impact on multiple object tracking performance. Other studies find that using bundled trajectories or smooth non-linear ones [15, 58] are more effective than straight ones in some specific conditions. CAST is designed to help users easily follow these guidelines with a simple and intuitive interface.

[1] Thompson et al.'s paper on animation design space [56] was published after we designed CAST.

2.3 Chart Animation Authoring Tool

A complete survey of visualization authoring tools is referred to Grammel et al. [21] and we focus on the authoring tools for chart animations, which can be classified into two categories: programming and non-programming tools.

Programming Tools. A few animation libraries [24, 42] have been developed for creating general animations, but the ones specifically for chart animations are scarce. One striking example is D3 [9], which provides a transition operator and a collection of interpolation functions for animating the charts made by D3 itself. By exploiting GPU hardware rendering, StarDust [41] produces animations with better performance while providing a similar API as D3. Such expressive low-level grammars facilitates the creation of highly-customized animations, however, the resulted verbose specification impedes rapid authoring. In contrast, gganimate [35] provides a high-level grammar for easily creating chart animations. However, it can only animate the charts created by ggplot2 [62] with limited types of animation transitions. By separating the chart animation form the creation step, the recently proposed high-level chart animation language, Canis [19], enables the concise, high-level specification of chart animations for the input of any data-enriched SVG chart. However, all these tools have a steep learning curve, especially for people who lack programming skills. To address this issue, CAST takes a visual specification approach for interactive chart animation specification by building and extending on top of Canis.

Non-Programming Tools. As discussed by Thompson et al. [56], there are three approaches for animation authoring: keyframing, procedural and template based animations. The procedural animations are mainly used for showing simulated processes, while most existing interactive authoring tools are based on templates. DataClips [4] allows non-experts to craft data videos by composing a sequence of predefined combinations of visualization types and animation types. Likewise, Adobe Stock [2] provides a set of data-driven motion graphics templates, while Flourish [28] further allows for integrating audio into chart animations. Such tools enable non-experts to create chart animations, however, their expressiveness is limited to the templates. Specifically, they support neither customizing the pacing or animation effects nor authoring animations for the visualizations beyond the predefined types.

In contrast, the keyframe-based tools, like Adobe After Effects [1], allow for precise control of visual properties and behaviors of visual elements with keyframes. Therefore, they are often used by experienced designers for crafting compelling chart animations. However, such tools do not provide data-driven abstractions of chart animations (e.g., their keyframes use absolute timing), resulting in a time-consuming and error-prone manual process to specify different chart states and each keyframe's timing. Inspired by Data Illustrator [27] that enhances graphic design tools with data encoding support for authoring expressive visualizations, our goal is to enhance traditional keyframe-based tools with visual specifications and data-driven auto-completion for authoring expressive chart animations.

2.4 Visual Specifications

The visual specification approach describes an abstract description of an object or a system by using a graphical vocabulary. As a result, it makes specification easier and more accessible with no need to know the details of how the system works [30]. The most notable example is Unified Modelling Language (UML) [47], which provides a variety of visual objects for modelling software systems.

Many visualization systems offer interactive visual specifications of visualization and analysis operations. Polaris [53] and its commercial successor Tableau [29] allow users to define visual specifications via drag-and-drop operations, namely, placing data fields onto "shelves" corresponding to visual encodings such as position, size, shape, or color. iVoLVER [32] provides an interactive visual language that enables users to acquire data from various sources and create interactive visualizations and animations. Lyra [48] allows visual specification of flexible custom visualizations such as force-directed layouts, trees and word clouds. Wrangler [25] takes a further step that can automatically suggest applicable data transformations based on the input visual specifications. Likewise, CAST also combines direct manipulation with automatic inference of relevant visual marks or keyframes, enabling designers to rapidly author desired chart animations.

2.5 Canis: CAST's Underlying Grammar

CAST is built on *Canis* [19], a high-level grammar to specify chart animations. Canis specifications can be used to describe a variety of animations of data-enriched SVG charts. Here, we briefly explain Canis because we refer to its core components while explaining the design and implementation of CAST. Please refer to the Canis website (https://chartanimation.github.io/canis) for the complete specifications and example animations. Later in Section 3.5, we will briefly explain how we extend Canis to enable interactive specifications of chart animations.

A Canis specification describes the animation of data-enriched SVG charts with a sequence of animation units *aniunits*. The input charts are embedded with the source data, and each *aniunit* is defined by a quadruple:

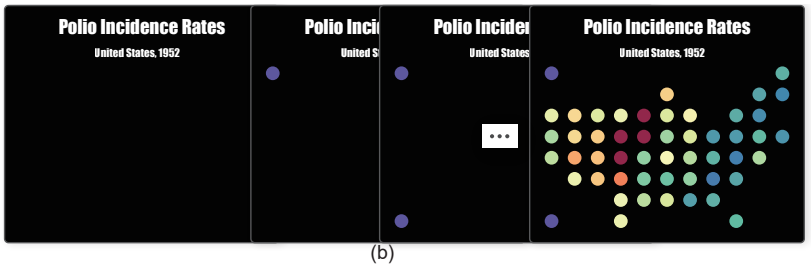
$$aniunit := (timing, selector, grouping, effects), \tag{1}$$

where the *selector* is used to select marks to be animated from the input charts using the W3C Selectors API [16]. The selected marks are further grouped into a set of elementary units that we refer to as *mark units*, with regard to categorical or nominal data attributes by *grouping*. The visual properties of the marks within the same mark unit update together during the animation. Note that *grouping* can be nested and thus a mark unit tree can be formed, where each level is grouped by one unique visually encoded data attribute.

Timing is defined both in *aniunit* and in each level of *grouping*, which controls the pacing of animation. It specifies when the animation starts using the reference (start with vs. after previous) and delay. While the *effect* component specifies the type of animation effect, easing function, and the duration of this effect. If not specified, Canis will automatically apply default values (e.g., default duration = 300ms).

```
"animations": [{
  "selector": ".title",
  "effects": [{ "type": "fade", "duration": 300 }]
}, {
  "reference": "start after previous",
  "selector": ".dot",
  "grouping": {
    "groupBy": "id", "delay": 100,
    "sort": { "field": "rate", "order": "ascending" }
  },
  "effects": [{ "type": "circle", "duration": 500 }]
}]
```

(a)



(b)

Figure 3: An example animation specified by Canis. (a) The Canis specification and (b) the resulted animation started with fading in the title, followed by the staggering animation of dots according to their associated data values in an ascending order.

Figure 3 (a) shows an example of Canis specification. It consists of two aniunits which describe the animations of the title and dots. In the dot unit, each dot is a mark unit, and it starts to animate a short time after the previous one started with the "circle" effect, according to their "rate" value in ascending order. The resulting animation is illustrated in Figure 3 (b).

3 CAST

In this section, we first present our design principles for CAST and introduce its visual specifications. We then describe its user interface and interactions along with three usage scenarios, and explain how CAST supports auto-completion for constructing keyframes and keyframe sequences. Finally, we provide two additional techniques employed for interactivity and better user experience, as well as implementation details.

3.1 Design Principles

With CAST, we aim to enable people who lack programming skills to easily create chart animations with a wide range of chart designs. To this end, we settled on the following three guiding design principles:

DP1: Provide explicit representations for chart animation specification. Being a general purpose animation authoring system, existing keyframe-based systems represent animations in an abstract manner within a purely timeline-based interface. For example, in Adobe After Effects and Adobe Premiere, a keyframe is represented as a diamond in a timeline. This abstract representation makes it nearly impossible for people to understand the animation they created without previewing. In contrast, to foster the understanding of specifications for chart animation, CAST uses explicit visual elements to represent animated marks (i.e., chart elements) and animation properties (e.g., timing, animation effects). Furthermore, CAST introduces an animation specification panel that blends storyboard with timeline. In their recent paper, which was published after we designed CAST, Thompson et al. [56] suggested that "storyboard interface and the timeline interface may be tightly coupled so that users can easily navigate between these two paradigms."

DP2: Support the easy construction of keyframe and keyframe sequences using data-driven inferences. The straightforward way to construct a keyframe is to manually select all of the necessary visual marks from the (data-enriched SVG) chart. However, this fully manual construction of keyframes is time-consuming and prone to errors, as well as even impractical as the number of keyframes grows. CAST strives to strike the right balance between the flexible graphics manipulation and procedural keyframe sequence generation based on Canis. It allows for efficiently constructing keyframes by suggesting possible visual marks based on the data attributes of the user-selected marks. CAST also suggests concrete and meaningful keyframe sequences based on the constructed keyframes.

DP3: Blend direct manipulation and configuration panel. Although direct manipulation is natural and intuitive, it is difficult to precisely specify quantitative values and perform repetitive actions [36]. CAST, as mentioned above, enables designers to directly manipulate visual elements to construct animations, whereas several relative timing properties cannot be exactly or easily specified. One example is that the given duration or delay time can hardly be exactly defined and another one is to manually specify meaningful animation order of keyframes, which is time consuming and error-prone. To overcome these issues, CAST consistently blend direct manipulation and several concise configurations panels together to specify such options that cannot be directly manipulated.

3.2 Visual Specifications

To improve the readability and understandability of the animation process **(DP1)**, we introduce visual specifications with four components—keyframe & keyframe group, timing, and effect—to visually convey the core aspects of animation.

Keyframe and Keyframe group. As described earlier in Section 2.5, a *mark unit* refers to a collection of marks whose visual properties are changed together during the animation. Each keyframe contains one such mark unit to be animated along with other units whose animations have been completed. These mark units can be grouped and nested, and the level of each group is indicated using the nested boxes with different shading intensities (mimicking the treemap [51]). These keyframes and keyframe groups convey meaningful steps in the animation, forming a storyboard. For example, as shown in Figure 4, the animation of a map incrementally revealing the polio incidence rates of the United States in 1952 can be represented with two keyframe groups: the

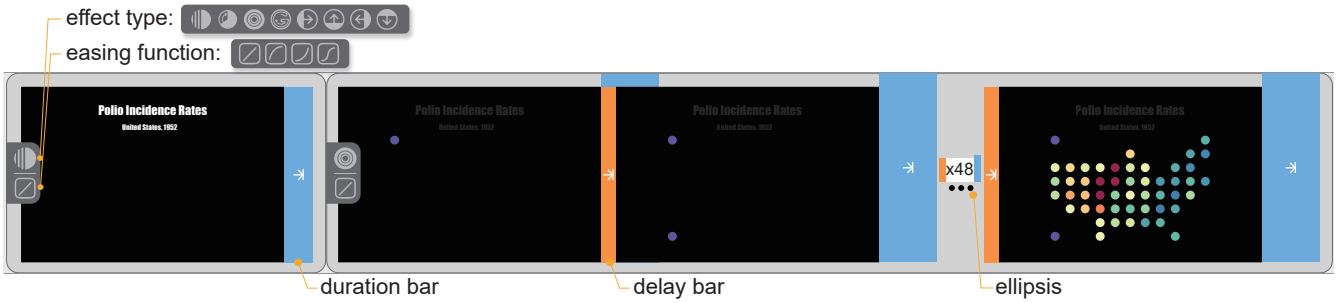


Figure 4: The visual representation of the animation of a map showing the polio incidence rates of the United States in 1952. The title fades in first, then all dots animate according to their rate in an ascending order with the "circle" effect. Such animation is represented by two keyframe groups with other animation properties including timing bars indicating duration and delay, and iconic representation of effect type and easing function (all effect types and easing functions are listed at the top left).

first group containing one keyframe to display the chart title and the second group having 51 keyframes (with 48 collapsed into an ellipsis) to show the corresponding zones one at a time.

Timing: Alignment, Duration, and Delay. The relative timing between keyframes is determined by three attributes—position, duration time, and delay time—in the storyboard-incorporated timeline (see the tracks within the animation specification panel in Figure 1 (d)). The horizontal placement of keyframes determines the order of their animations. The adjacent keyframes play sequentially, while the keyframes vertically aligned across multiple tracks play simultaneously (vertical alignment is represented with a dark-gray dotted line)[2]: Figure 1 shows an example of the alignment between labels (corresponding to dots) and links (connecting the dots).

CAST conveys the timing properties of each keyframe—duration and delay–using timing bars. The bar width represents the time length and bar color encodes the type (blue for duration and orange for delay). The duration bar is placed on the right side of each keyframe, while the delay bar is placed on the left side of a keyframe to show the delay *before* the corresponding animation starts. (As a keyframe group is a logical collection of keyframes, it does not have a duration bar: its duration is determined by the duration of keyframes that belong to this group.)

CAST allows the next keyframe to start before the previous keyframe finishes. In this case, the delay bar of the next keyframe will be aligned to the left with the duration bar of the previous keyframe. To avoid the complete occlusion between them, the duration bar is scaled in Y direction to be a bit taller than the delay bar. For example, in the second keyframe group in Figure 4, the delay bar of the second keyframe is left-aligned with the duration bar of the first keyframe, conveying that the animation for the second keyframe will start while the first keyframe is still animating.

All keyframes in the same keyframe group share animation properties, and thus showing all keyframes in a keyframe group is redundant, consuming a lot of screen space. Therefore, CAST shows only three (first, second, and last)[3] keyframes in one keyframe group and collapses keyframes under the ellipsis (Figure 4). The number of the collapsed keyframes will be labeled on the ellipsis.

Animation Effects. We use icons to represent two properties of animation effects—effect type and easing function—as they are categorical variables. CAST currently provides eight types of effects (e.g., fade, wipe) and four types of easing (Figure 4) drawn from commonly used tools like PowerPoint. We note that it is straightforward to add additional translation effects and easing functions. Since all keyframes in the same topmost keyframe group share the animation effect, the effect type and easing function are only shown on the topmost keyframe group.

3.3 User Interface and Interaction

The CAST user interface consists of a data panel, a chart panel, an animation preview panel, and an animation specification panel (Figure 1). We explain CAST's interaction using three example scenarios (**open this PDF in Acrobat Reader to view the animations** shown in each scenario). To see how CAST works, please refer to the gallery videos on the CAST website (https://chartanimation.github.io/cast/).

Interaction Mechanisms. When the data-enriched SVG chart is loaded[4], it is assigned to the default animation which is to fade in the entire chart: the corresponding representation of a single keyframe with all visual elements in the chart as a mark unit is generated on the animation specification panel. To author the customized animation for the chart, the author can start from creating the mark unit to be animated by simply selecting the desired marks on the chart panel. The data table and the input chart are tightly coupled: the author can select the marks from the data table when the marks need to be selected based on the data values that are not clearly displayed (for example, see Scenario 2). Moreover, when it is difficult to select marks in a few selection operations, (e.g., selecting all blue

[2]Keyframes and keyframe groups are equivalent in many aspects. For the sake of simplicity, we describe only with keyframes and highlight the cases where keyframes and keyframe groups behave differently.

[3]CAST shows only two (first and last) when zoomed out to see a high-level overview of the animation, as illustrated in Figure 9.

[4]The data-enriched SVG chart can be automatically generated with our online generator (https://chartanimation.github.io/canis/marker/index.html) by using the SVG charts created either with interactive authoring tools like Charticulator or using a programming library like D3.

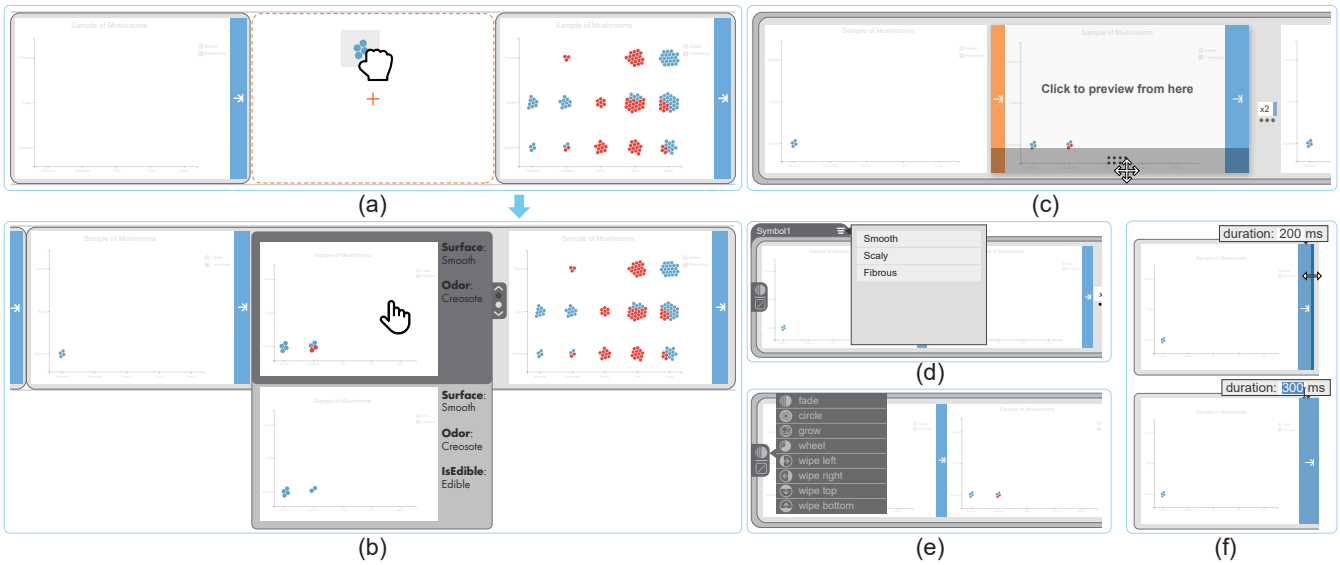


Figure 5: An example illustrating the keyframe and keyframe sequence construction procedure. (a) a mark unit with four blue dots is dragged over the dropzone ahead of the keyframe group containing all dots; (b) after creating a keyframe for the selected dots, a list of possible next keyframes is suggested to let users select to complete the keyframe sequence; (c) dragging one keyframe to make keyframes animate a short time after the previous one started; (d) rearranging the animation order of sibling keyframe groups; (e) selecting effect type for keyframe group; and (f) two ways to update the duration length.

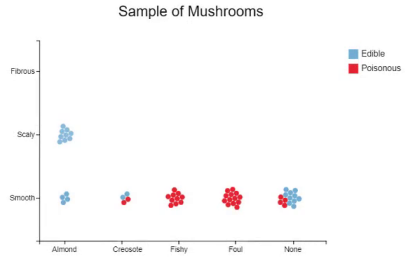
dots in the faceted dot plot in Figure 2 (d)), the author can first select a small portion of the desired marks as an example, then they can either select the mark unit which is automatically recommended by CAST or keep selecting from the input chart manually.

To generate a keyframe, the author can drag and drop the selected marks on the animation specification panel (see Figure 5 (a)). Then CAST automatically generates a keyframe for the corresponding animation unit and suggests a list of potential next keyframes for the author to select (see Figure 5 (b)). This suggestion-selection process will proceed recursively until the keyframe sequence is determined.

Once the keyframe sequence is generated, the author can rearrange the animation order of the sibling keyframe groups using the popup panel on their parent keyframe group (Figure 5 (d)). To change the timing of keyframes, the author can drag one keyframe and adjust its relative position to the previous one. CAST will hint the changes to be made while the keyframe is being dragged (Figure 5 (c)). As for editing the length of duration or delay, the author can change the width of the timing bar by dragging the end of it. To specify precise duration, they can enter the number in the popup input box shown when hovering the timing bar (Figure 5 (f)). Finally, the author can specify the desired animation effect and easing function by selecting from the callout list on the top-most level keyframe group (Figure 5 (e)). CAST also supports undo/redo operations which enables the author to recover from unintended modifications to the animation.

The change on any keyframe will be automatically applied to all keyframes within the same top-most level keyframe group. The result animation can be previewed on the animation preview panel using the media controllers. (Clicking on a keyframe will start the preview from the keyframe.)

Scenario 1: Animation of the faceted dot plot.

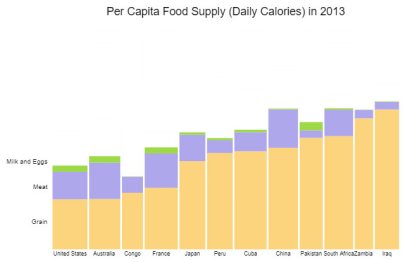


We will create the animation with a faceted dot plot (Figure 2 (d)), as illustrated in the inset figure above. The chart depicts 170 samples of mushrooms, grouping them by their odor (x-axis) and surface quality (y-axis). Each dot in the chart corresponds to one mushroom sample, where its color indicates whether it is poisonous or not. The animation of this chart starts with the title, axis, and legend fading in together. Then the dots in each cell appear together from the cell on the bottom left to the top right one after another. Meanwhile, there is a short pause between animation of cells within the same row and a longer pause between rows.

To first animate the title, axis, and legend, we start by selecting and dragging them to the animation specification panel to create the first keyframe. To specify the dots in the first cell to animate, we create another keyframe with the four dots on the bottom left corner. Then CAST provides a list of next keyframes indicating all

possible keyframe sequences, where each one corresponds to the unique partition strategy for all dots. We choose the one showing the keyframe sequence we wanted, which was also labeled as to partition by Surface first, then partition by `Odor` (see Figure 5 (b)). Finally, to add delay between keyframes, we adjust the distance between two adjacent keyframes by dragging one away from the other. After inserting delay between the keyframe groups in the same manner, we increase the delay by stretching the right border of the delay bar.

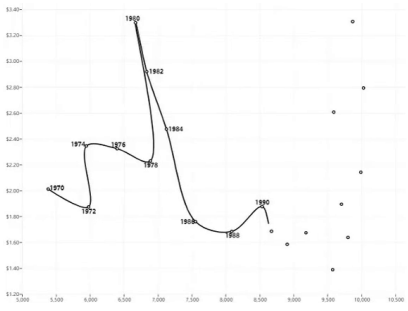
Scenario 2: Animation of the Mekko Chart.



We will create an accumulation animation with a Mekko chart, which shows per capita food supply. In the chart, each column represents the proportion of calories provided by different foods in one country, and the color of each rectangle encodes the type of the food. The animation starts by fading in the title and axes, and then the labels of food types and corresponding rectangles are interlaced to animate with fade and wipe effects, respectively.

To first fade in title and axes, we drag them from the chart panel to the animation specification panel to create the first keyframe. To animate rectangles by food type, we first select the rectangles corresponding to `Grain` to create another keyframe. In response, CAST automatically completes the animation of accumulating rectangles of other types of foods since there is only one possible animation sequence. After that, we change the effect type of rectangles to "wipe bottom." To let the labels fade in before the corresponding rectangles, we first create the staggering animation of labels in the same manner, then we drag the keyframe group of rectangles to align it with the labels on the element level.

Scenario 3: Animation of the Connected Scatterplot.



We will create an animation with Hannah Fairfield's connected scatterplot [17] (Figure 2 (b)), which shows the relationship between driving distance and oil price over time. In this chart, each dot representing the oil data of a year is positioned according to miles driven (x-axis) and its price (y-axis), having year as a text label. Two dots for adjacent years are connected by a link. The animation starts with both axes fading in followed by all dots fading in together. Then each link grows in chronological order, and the year label fades in once the link reaches the corresponding dot.

To create the first keyframe of the axis, we select several axis ticks with their labels from the input chart. Then CAST recommends the entire axis including ticks, tick labels, and grids. We drag them to the animation specification panel to create the first keyframe. Then, we create the second keyframe with all dots in a similar way. In response, CAST automatically creates three keyframe groups each containing a single keyframe. These keyframe groups, placed horizontally adjacent to each other, depict the animation of three types of marks: dots, labels, and links with the default effect "fade in," respectively.

To make the labels fade in one by one, we drag the first label from the chart panel to the dropzone ahead of the keyframe group corresponding to labels. To make the labels fade in when the link reaches the corresponding dot, we drag the keyframe group of links and align it with the keyframe group of labels on the element level. Finally, we change the effect type of links animation to "grow" and extend their duration.

3.4 Auto-completion of Keyframes and Keyframe Sequences

In this section, we explain how CAST infers suggestions to support auto-completion for constructing both keyframes and keyframe sequences (DP2).

Mark Unit Recommendation. Given the input chart (Figure 6 (a)), CAST infers the underlying mark units based on the selected visual marks (Figure 6 (b)) and recommends all marks in the corresponding unit for completing the selection. To convey meaningful data patterns, the mark unit is obtained by analyzing the visually encoded data attributes of the selected marks. Specifically, all marks in each mark unit should have the same values of the encoded attributes S, while these marks can have different values of the encoded attributes D. For example, S only has the attribute `IsEdible` in Figure 6 (b), while D includes `Odor` and `Surface`. Accordingly, all un-selected marks in the mark unit associated with the same data value as the selected marks are suggested. In Figure 6 (c), two mark units are formed corresponding to two different values of the attribute `IsEdible` and all blue marks are highlighted and suggested for selection while the others are translucent.

In case that D contains several attributes encoded by different channels, the suggested mark units might not meet with users' expectation. For example, two selected points in Figure 6 (d) have different values in three attributes and thus one point in each group in which all marks have the same values in three attributes is suggested (Figure 6 (e)). However, the suggested two points in different colors from each of the cells containing both blue and red classes might not match user intention, because the selected marks do not reflect such preference. To address this issue, CAST filters D with the effectiveness principle of visual encoding [34] and only keeps those encoded with the most effective channel. In Figure 6 (d), the attributes `Odor` and `Surface` are more important than `IsEdible` since they are encoded by the most effective channel, position. Hence, the attribute `IsEdible` does not need to be considered and one point in each cell is suggested to be selected (see Figure 6 (e)). Note that if the suggestion is not accepted, the authors can select

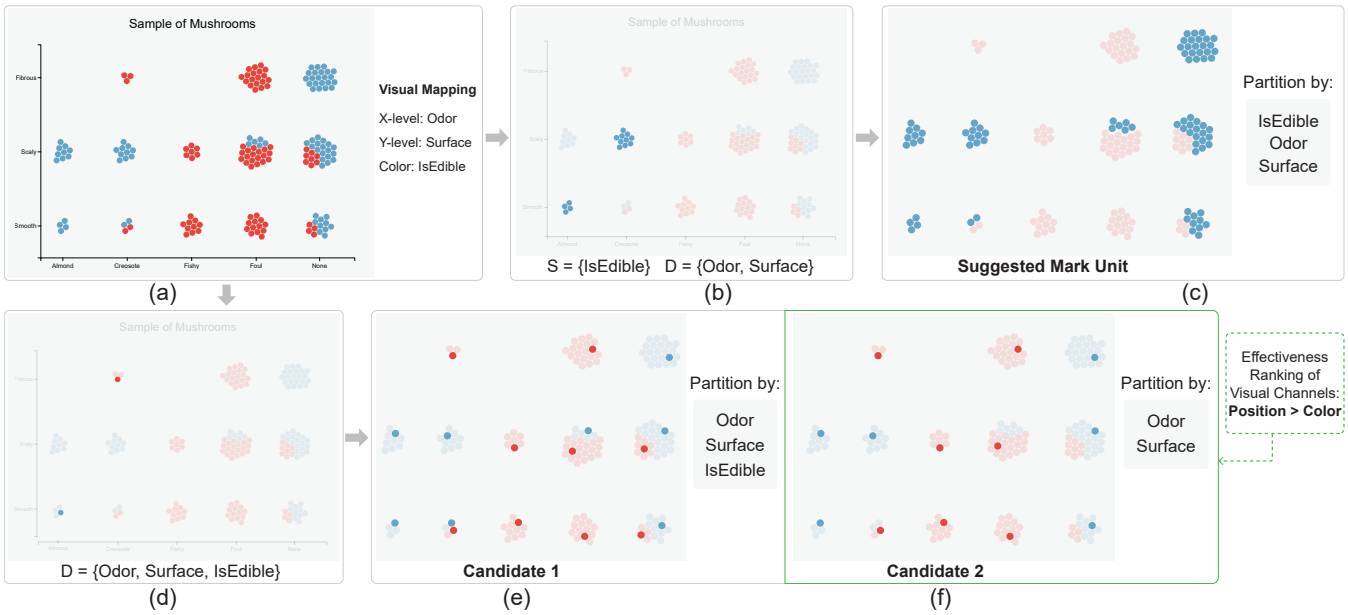


Figure 6: CAST suggests additional marks to be selected based on the user selected marks from (a) the input chart. When (b) two cells with only blue dots are selected, all blue dots in every cell are highlighted as (c) the suggestion result. When (d) two different colored dots from two different cells are selected, from (e, f) the two candidates our system recommends the (f) second candidate based on the effectiveness ranking.

additional marks to let CAST keep recommending new mark units, or turn off the suggestion mode and re-select marks.

Next Keyframe Recommendation. Once the keyframe with the selected mark unit is generated on the animation specification panel, CAST automatically infers the possible animation orderings of all such units and suggests a list of unique next keyframes for the author to select to complete the whole keyframe sequence.

To generate meaningful sequences, CAST hierarchically groups the mark units internally based on the visually encoded data attributes. For mark units with a set of visually encoded data attributes, CAST provide diverse animation options by iteratively permuting and filtering data attributes. Give the set of attributes A, CAST first generates multiple ordering schemes corresponding to all permutations of attributes and then filters out data attributes encoded by the relatively least effective visual channel from A. Next, CAST re-do this process until A is empty. Figure 7 shows an example, where the points in the selected mark unit have three visually encoded attributes: `Odor`, `Surface`, and `IsEdible` (see Figure 7 (a)). Since they are encoded by two position channels (x-level and y-level) and the color channel, respectively, there are 8 possible ordering schemes (6 from permutations of all three attributes, and 2 from permutations of the two attributes encoded by the position channel). Figure 7 (b-d) show three of those animation orderings.

Based on all possible keyframe sequences, CAST suggests all unique second keyframes as the next possible keyframes. By checking the eight ordering sequences of the example in Figure 7, there are two unique possible second keyframes (highlighted with purple border). If the one in Figure 7 (e) is selected, the whole sequence is determined; otherwise, multiple third keyframes are suggested (highlighted with pink border).

Note that if the mark unit is not data-encoded, there is only one keyframe sequence where the ordering is determined by the index of mark unit. For example, if the first tick and label of the axis are selected, CAST generates the sequence of animating pairs of ticks and labels one by one.

3.5 Incremental Compiling, Semantic Zooming, and Implementation Details

Incremental Compiling. With original Canis, any small changes lead to recompiling the entire specification. This might become too slow to achieve interactivity when authoring complex or expressive animations. To address this issue, we extended Canis with an incremental compilation [37] that re-compiles only a portion of the specification affected by modifications. Given a modified specification resulted by user interaction, we first use the diff mechanism to quickly locate its difference with the previous version and generate the *changeset*. Then, we apply the build-bind-evaluate operator to update the mark-unit tree in terms of the *changeset*. Specifically, the build operator locally updates the tree structure in terms of the new grouping specification. Once the tree is updated, the bind and evaluate operators update the effect and timing properties of each mark unit. An example is shown in Figure 8, where the two-level mark-unit tree (see Figure 8 (b)) is updated to a three-level one (see Figure 8 (d)).

Semantic Zooming. Because our keyframes show a thumbnail of the chart (to fulfill DP1), the animation sequence in the specification

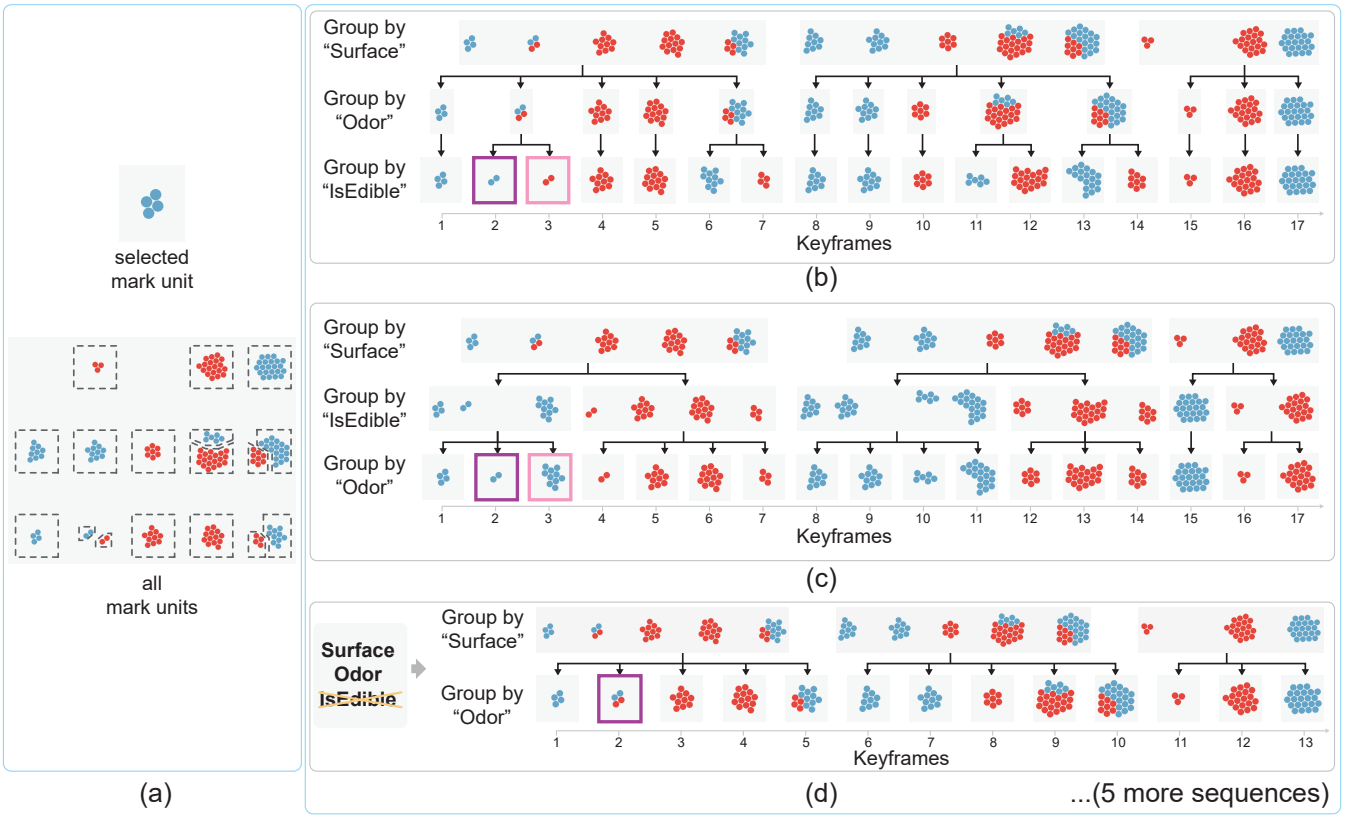


Figure 7: CAST computes possible keyframe sequences by analyzing the visually encoded data attributes associated with the selected mark unit. (a) the selected mark unit (top) and all mark units (bottom); (b, c, d) three out of eight possible keyframe sequences generated by grouping all mark units with different permutations of data attributes, where (d) is generated with two attributes after removing `IsEdible` which is encoded by a less effective visual channel, color. The highlighted keyframes in (b, c, d) are the keyframes to be suggested to the users (after deduplication).

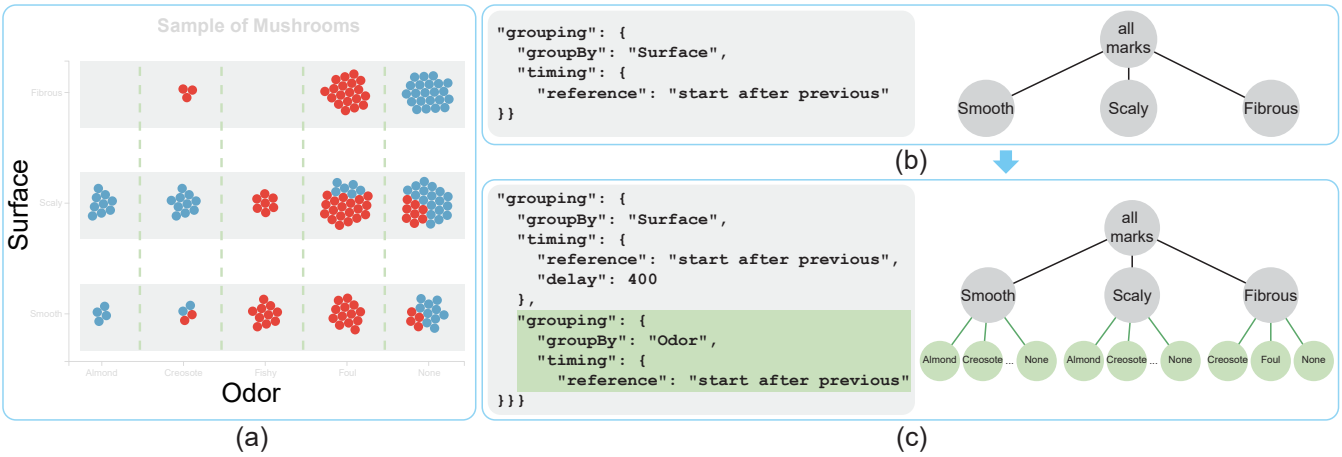


Figure 8: The illustration of an incremental compilation for the modified specification with the input chart in (a). (b) The original specification (left) and the resulted two-level mark-unit tree (right); (c) the modified specification where the changed part is highlighted (left) and the updated three-level mark-unit tree (right).

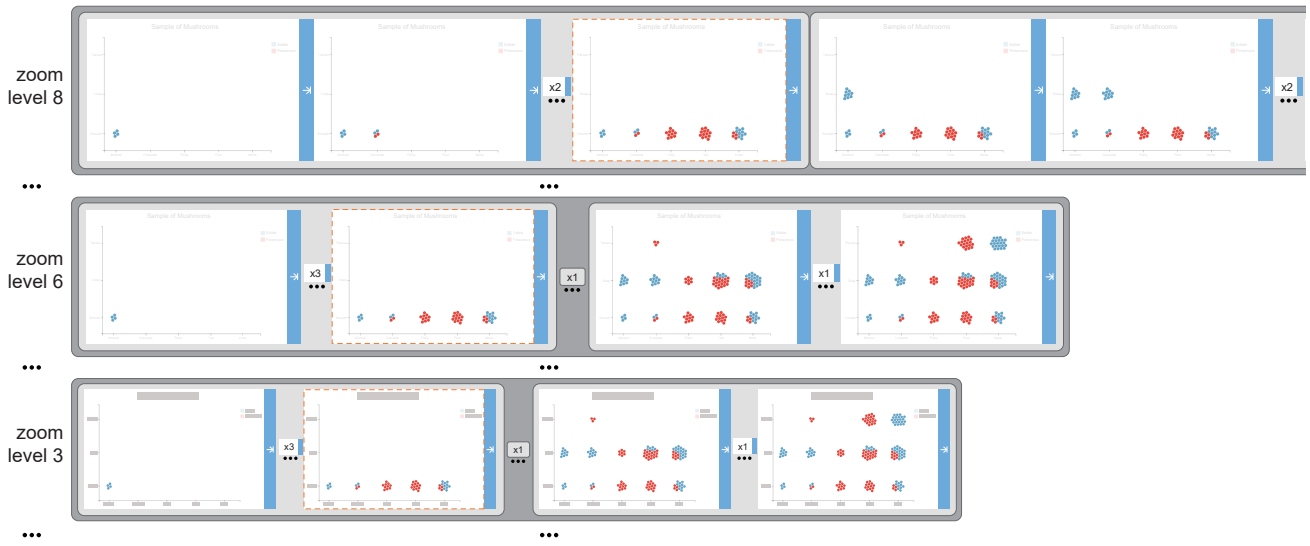


Figure 9: An example illustrating how the keyframe sequence shown in Figure 7 (d) is changed with semantic zooming on different zoom levels. We present this keyframe sequence on three zoom levels since it shows the same number of keyframes or keyframe groups under level 6, while the size is scaled; The last keyframe in the first keyframe group is highlighted with an orange frame to illustrate the changes when zooming.

panel can become too long to be easily accessed. To address this issue, we employ semantic zooming [8] in the animation specification panel with two design goals: (1) preserve the entire hierarchical grouping structures of keyframe sequences and (2) maintain the visibility of visual marks when zooming out.

To achieve these goals, CAST sets l zoom levels where the sizes and numbers of keyframes and keyframe groups are gradually reduced with the same scaling factor. At level l, each keyframe group consists of three child components: the first, second, and last keyframe groups or keyframes at the next level (see the keyframe sequence at zoom level 8 in Figure 9) and the second keyframes in each keyframe group are removed at level $l-1$. As the zooming level decreases by one, the mark unit tree is traversed one level up, and the second keyframe group of the sibling groups at the corresponding level is removed, see the sequences of the zoom levels 6 in Figure 9.

When the zooming level is less than $l/2$, the mark appearance is modulated with two strategies to improve visibility (R2). First, all 2D visual marks are enlarged a little after being shrunk with the scale factor. For example, the area of the closed marks (e.g., circles) or the path related marks (e.g., links) is enlarged by increasing the radius or thickness with $(l/2-l)*2$ pixels. Second, the text related marks (label, title and legend) become hardly to be read and thus we generate the bounding boxes of them to indicate the existence of them. In doing so, the bounding boxes provide navigational assistance while directing users to focus on data-encoded visual marks. In our experiment, we found that $l=8$ is enough in most cases since the depth of the mark unit tree in most animation is less than 5.

Implementation Details. CAST is an HTML5 web application implemented with TypeScript and Webpack, following the Redux architecture [52]. It maintains the *State*, which records the status of all components in the system. Specifically, the status of the animation specification panel is described using a *Canis Specification*. The access and alteration of the State are controlled by the "Store" part of Redux. For each editing interaction on this panel, an "Action" as the payload carrying the update information will be emitted to the Store and then the store alters the Canis specification in the state according to the received information. Once the state is successfully updated, the renderer will be invoked by the store to update components on the animation specification panel. The other components in CAST share the same mechanism.

4 EVALUATION: GALLERY AND USER STUDY

To demonstrate the expressive power of CAST, we created a wide variety of chart animations with a diverse collection of charts and animation effects. Fig. 2 shows eight examples from our gallery and the full gallery can be found in the website, along with animations, detailed descriptions, and videos illustrating the creation processes.

The visualization research community has acknowledged the challenges and limitations of comparative studies for evaluating visualization authoring systems (especially designed for communication purposes) [40, 49]. We face similar issues: there are no prior works that allow people to easily create chart animations. In addition, as CAST introduces a new visual specification, the authors first need to learn and understand the specification to use CAST for authoring chart animations. We therefore decided not to conduct a formal comparative study, but we instead designed a study to evaluate both the visual specification and the system as described below.

4.1 Study Design

Our user study consists of two parts. **Part 1 (Understanding Animation Specification)** assesses if people can learn and understand CAST's visual specification. **Part 2 (Reproducing Chart Animations)** determines if people can reproduce customized animations using CAST, following a similar methodology used for evaluating chart authoring tools [27, 38, 39, 64, 66].

Participants. We recruited 18 participants (6 females; 12 males) from the local university. Their ages range from 22 to 32, with an average age of 25. All participants have a normal or corrected-to-normal vision. They major in computer science and most of them have the experience of using video or animation editors, such as TechSmith Camtasia [54], Adobe After Effect [1], iMovie [5], and Movie Maker [33].

Apparatus. We used a desktop computer with an Intel i7-8700K 3.7GHz CPU and 16GB RAM, using two 27-inch LCD displays placed side-by-side, both running at 1920×1080 resolution. In part 1, we only used one display and recorded participants' choices for each task. In part 2, the input chart, target animation, and animation description were shown on the left monitor, and asked the participants to reproduce animations on the right monitor. During this procedure, we logged all of the participant's interactions with CAST, recording the *time* taken by participants for completing each task. We also captured the whole reproduction process using a screen recorder.

Tasks. We prepared three tasks (Figures 2 (a-c)) for part 1, covering visual elements and layouts used in the visual specifications, and four animation reproduction tasks (Figures 2 (d-g)) for part 2, encompassing basic interactions in CAST. For both parts, the expressiveness and complexity of the subsequent tasks gradually increased.

Procedure. After briefly introducing the study goals and procedure, we asked participants to fill out a pre-study questionnaire. We then provided the tutorial of how to read the visual specification with three animations. After completing three training tasks to expand the understanding of the visual specifications, participants can click the "Start Task" button to perform part 1 of the study. We asked participants to verbally describe the animation from the visual specification, so that we can ensure that they understood it. Finally, we asked participants to complete a questionnaire asking about the visual specifications in terms of easy to learn and easy to understand by using a 5-point Likert scale with 1 being "Strongly Disagree" and 5 being "Strongly Agree."

After a 10-min break, the participants proceeded to part 2. We first taught the participants how to use CAST by demonstrating the basic features with two animations. We then asked the participants to complete four training tasks to familiarize themselves with the system. The participants then performed the four reproduction tasks. Before they started each task, in addition to showing the input visualization chart and target animation, we provided the description of key animation features including the duration and delay, effect type, and easing function because these features are not easily recognizable from the video. We also asked the participants to describe the animation to us to ensure that they understood the task. When they were ready, they could click the "Load Chart" button to load the input chart in CAST and start the task. During the reproduction process, we asked participants to think aloud about their experience with CAST, and provided hints to the participants when they asked for help. After participants finished all four tasks, we asked them to rate CAST on four criteria using a 5-point Likert scale, with 1 being "Strongly Disagree" and 5 being "Strongly Agree." On average, the entire session lasted about 80 minutes (30 for Part 1, 40 for Part 2, and 10 for a break).

4.2 User Study Results

Understanding Animation Specification. Fifteen (out of 18) participants found the correct animation in all three tasks without any hints. Two participants (P15 and P17) made an incorrect choice in Task 1: they interpreted that the keyframe sequence is grouped first by month and then by mortality, but it is grouped by mortality first. One participant (P3) was confused with the relative timing between keyframes and keyframe groups in task 2. As shown in Fig. 10(a), the responses from the after-study questionnaire indicate that the visual elements used in the visual specifications are easy to learn (*Avg* = 4.7) and easy to understand (*Avg* = 4.6).

Reproducing Chart Animations. By carefully checking the reproduced animations in part 2, we found that 17 out of 18 participants successfully reproduced the target animations in all four tasks. Nine participants completed them without any hints, and eight participants needed only a few hints. The one remaining participant failed to reproduce Task 4, forgetting how to further partition an existing keyframe sequence and rearrange the order of keyframe sequences of different marks.

The average task completion time was less than two minutes except for Task 4 (282 seconds); see Figure 10 (b) for the summary. Note that we did not explicitly ask the participants to complete the task as quickly as possible. With regards to subjective ratings, as shown in Figure 10 (a), participants indicated that CAST is easy to learn (*Avg* = 4.5) and use (*Avg* = 4.7), and that they can efficiently create chart animations (*Avg* = 4.7) and enjoy the tool (*Avg* = 4.6). Our participants were generally positive about various features of CAST. Participants appreciated the direct manipulation of relative timing between keyframes or keyframe groups. Six participants mentioned that the auto-completion of the keyframe sequence impressed them most.

However, we also identified some usability issues of CAST. The areas and affordances for dragging of the keyframe and keyframe group are quite different: drag bar on the bottom for the keyframe and the tab on the top left corner for the keyframe group. As they do not clearly convey how they can be interact with, participants were sometimes confused about which one should be dragged (keyframe or keyframe group) and to where, especially when there is only one keyframe in the keyframe group. In addition, sometimes during a component is dragged, multiple signals might be emitted to the system. For example, dragging a keyframe group to cross multiple keyframe groups may send several timing update signals to the system. This may cause some unintended results (e.g., losing focus of the cursor to the dragging component). We already revised the signal management strategy to address the second issue, and plan to improve our interaction design to resolve the first issue.

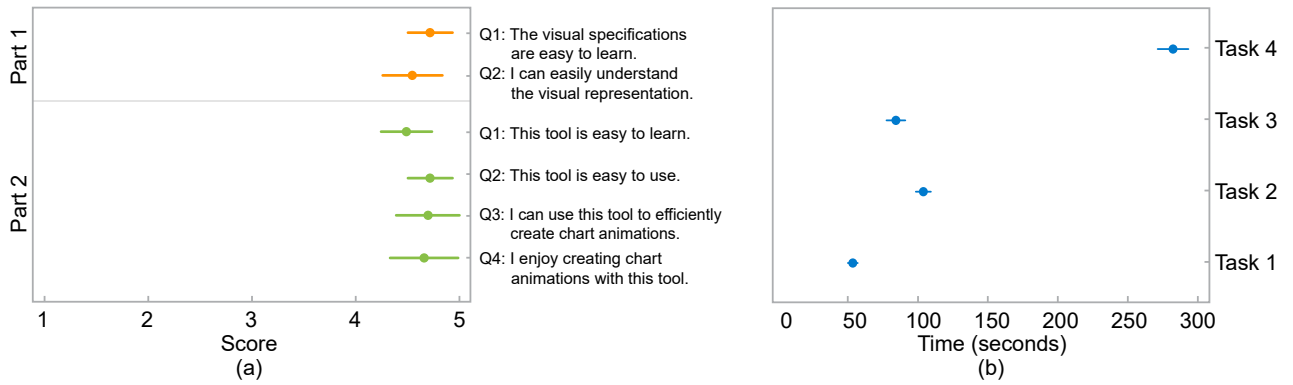


Figure 10: Results of the user study. (a) The 95% confidence interval in scores to two satisfaction questions in part 1 and four questions in part 2: the higher score the better. (b) The completion time in seconds for four reproduction tasks in part 2: error bars represent the standard deviations.

5 DISCUSSION AND FUTURE WORK

General Reactions. The encouraging results—task completion time and subjective rations—of our user study indicate that the design of the visual specification and CAST facilitate the easy creation of chart animations. In general, our study participants also expressed excitement about the expressive animations CAST enabled. P18, for example, was particularly satisfied with the animation he created with CAST, and remarked,

> *"I was attracted by the animations at the first sight, since I never thought those charts can be animated in such expressive manner. Meanwhile I was worried that I would spend a lot of time and effort on authoring those animations. However, after only a few steps of playing around the graphics, the animation which just amazed me is now created by myself."*

Participants appreciated the direct manipulation and the auto-completion of the keyframe sequence CAST offered. For example, P4, who is interested in designing visual analytic systems, commented,

> *"The suggestion on mark selection and sequence construction impressed me most, it provided me with multiple possible animations and simplified my authoring procedure. And the direct visual manipulation and system feedback on the graphics also ensures the consistency of my interaction with the system."*

Furthermore, the expressiveness of the resulted animations combined with the simple yet interesting graphical interactions seemed to make the animation authoring process a fun activity. For example, P10 noted, *"The authoring process is like playing a game, I can get such a fancy animation while having fun."*

Expressiveness of CAST. Because CAST supports data-driven animations, it can cover a wide range of charts including highly expressive ones: the examples in our galley leveraged charts created with Charticualtor [39] and D3 [9]. In terms of the types of animation (using DataClip's taxonomy [4]), CAST currently supports four types of animations: *creation*, *deconstruction*, *accumulation*, and *transition* even though most animations used in the paper can be categorized as the *creation* and *accumulation*. (Please refer to the example gallery of CAST at https://chartanimation.github.io/cast). We note that CAST disables some interactions for the authoring of *transition* animations. For example, the keyframe can no longer be dragged to specify the mark units in one chart to be animated successively since it might result in invalid visual mappings during the transition between multiple charts. CAST cannot produce all animations that can be created using Canis because of the limitations in the visual specifications. The visual specification does not provide components for two operators defined in Canis: (1) it does not allow users to bind data attributes to the animation duration and delay yet and (2) it does not support multi-view animations, which facilitate a side-by-side comparison. Addressing these issues will require us to carefully extend our visual specifications and user interface. We also note that it may require further research to design an interactive system to support additional animation types from the DataClip's taxonomy like *annotation* or *embellished*.

CAST also inherits a limitation of Canis which is lacking the support of staged transitions. As CAST builds on Canis, we plan to extend Canis by enabling the specification of intermediate key steps in transition and explore the possibility of auto-completing the transition animation by suggesting the intermediate keyframes based on prior perceptual studies [23]. Meanwhile, we will enrich our visual specifications with components for representing transitions and refine CAST's user interface for interactively defining such intermediate steps.

Study Limitations. Our study participants were all students majoring in computer science. Even though their daily job does not necessarily involve creating chart animations (or even visualizations), they are presumed to be able to perform computational thinking. It could be helpful to conduct further studies to assess if people with different backgrounds (e.g., design, marketing) can also create expressive chart animations with CAST.

Our user study employed a reproduction study [40], following the approach commonly used for evaluating visualization authoring systems, including Data Illustrator [27], DataInk [64], and Charticulator [39]. We assessed if people could produce chart animations using CAST when provided with a reference animation and a short training (in the second part of our study, the tutorial and training

took about 25 minutes). Furthermore, CAST requires a chart design to craft animations. With our study, which inherits the limitation of this study methodology, we cannot conclude if people will be able to create expressive animations using CAST. In the future, we hope to evaluate CAST's expressiveness with designers in a hackathon setting, allowing enough time to think about creative animations. In addition, as our system is deployed on the web, we want to collect chart animations people create with CAST in the future, while logging people's usage.

We also note that our work can be complimented with future studies. For example, it will be informative to conduct a study focusing on auto-completion as a technique and investigate how it can help users create chart animations. In addition, in-depth studies on what animation types and effects are most challenging, contribute the most to expressivity, and require the use of auto-completion.

To design our questionnaire, we also referred to the questionnaires used in previous reproduction studies to evaluate chart authoring tools [40]. However, the positive framing of questionnaire statements (e.g., the visual specifications are easy to learn) might have biased participants to give a higher rating [65]. In the future studies, we will use statements framed more neutrally (e.g., To what extent do you feel that the visual specifications are easy / difficult to learn) to avoid framing influence on participants' subjective ratings.

6 CONCLUSION

Despite the effectiveness and popularity of chart animation, it is not easy for people without programming skills to craft one, especially with bespoke charts: this is partly due to the lack of authoring tools specifically designed for chart animations. In this paper, we presented CAST, an interactive chart animation authoring tool that helps people easily produce expressive animations using a wide variety of chart designs. Unlike most existing timeline-based animation authoring tools, CAST introduces the visual specifications specifically designed for chart animations to foster easy understanding of the animation process. It also leverages the direct graphical manipulation of the visual elements and auto-completion of keyframe and keyframe sequences to facilitate the easy construction of animations. We explained the design principles of CAST to achieve our goal in help people who lack programming skills to easily create chart animations. With three usage scenarios, we illustrated how the visual specification and interface of CAST enables rapid animation creation process. We also explained in detail about the auto-completion based on data-driven inferences which improves the authoring efficiency. Through a two-part user study, we assessed the learnability and understandability of the visual specifications, and CAST's learnability and usability. CAST is available at https://chartanimation.github.io/cast, along with our example gallery that demonstrates the system's expressiveness.

ACKNOWLEDGMENTS

This work was supported in part by the grants of the NSFC (61772315, 61861136012), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No.VRLAB2020C08), and the CAS grant (GJHZ1862).

REFERENCES

[1] Adobe After Effects cc. 2021. Adobe After effects. https://www.adobe.com/ca/products/aftereffects.html. [Online; accessed 5-September-2020].
[2] Adobe Stock. 2021. Adobe Stock. https://stock.adobe.com. [Online; accessed 5-September-2020].
[3] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Christophe Hurter, and Pourang Irani. 2015. Understanding data videos: Looking at narrative visualization through the cinematography lens. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*. 1459–1468. https://doi.org/10.1145/2702123.2702431
[4] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Andres Monroy-Hernandez, and Pourang Irani. 2016. Authoring data-driven videos with dataclips. *IEEE Transactions Visualization and Computer Graphics* 23, 1 (2016), 501–510. https://doi.org/10.1109/TVCG.2016.2598647
[5] Apple. 2021. iMovie. https://www.apple.com/imovie/. [Online; accessed 5-September-2020].
[6] Daniel Archambault, Helen Purchase, and Bruno Pinaud. 2010. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions Visualization and Computer Graphics* 17, 4 (2010), 539–552. https://doi.org/10.1109/TVCG.2010.78
[7] Lyn Bartram and Colin Ware. 2002. Filtering and brushing with motion. *Information Visualization* 1, 1 (2002), 66–79. https://doi.org/10.1057/palgrave/ivs/9500005
[8] Benjamin B Bederson and James D Hollan. 1994. Pad++ a zooming graphical interface for exploring alternate interface physics. In *Proc. ACM symposium on User Interface Software and Technology*. 17–26. https://doi.org/10.1145/192426.192435
[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE Transactions Visualization and Computer Graphics* 17, 12 (2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185
[10] 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 Visualization and Computer Graphics* 26, 1 (2019), 364–374. https://doi.org/10.1109/TVCG.2019.2934397
[11] Amira Chalbi, Jacob Ritchie, Deokgun Park, Jungu Choi, Nicolas Roussel, Niklas Elmqvist, and Fanny Chevalier. 2019. Common Fate for Animated Transitions in Visualization. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 386–396. https://doi.org/10.1109/tvcg.2019.2934288
[12] Fanny Chevalier, Pierre Dragicevic, and Steven Franconeri. 2014. The not-so-staggering effect of staggered animated transitions on visual tracking. *IEEE Transactions Visualization and Computer Graphics* 20, 12 (2014), 2241–2250. https://doi.org/10.1109/TVCG.2014.2346424
[13] Fanny Chevalier, Nathalie Henry Riche, Catherine Plaisant, Amira Chalbi, and Christophe Hurter. 2016. Animations 25 years later: New roles and opportunities. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. 280–287.
[14] Pierre Dragicevic, Anastasia Bezerianos, Waqas Javed, Niklas Elmqvist, and Jean-Daniel Fekete. 2011. Temporal distortion for animated transitions. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*. 2009–2018. https://doi.org/10.1145/1978942.1979233
[15] Fan Du, Nan Cao, Jian Zhao, and Yu-Ru Lin. 2015. Trajectory bundling for animated transitions. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*. 289–298. https://doi.org/10.1145/2702123.2702476
[16] J David Eisenberg and Amelia Bellamy-Royds. 2014. *SVG Essentials: Producing Scalable Vector Graphics with XML*. " O'Reilly Media, Inc.".
[17] Hannah Fairfield. 2009. Driving Shifts Into Reverse. https://archive.nytimes.com/www.nytimes.com/imagepages/2010/05/02/business/02metrics.html
[18] Danyel Fisher. 2010. Animation for visualization: opportunities and drawbacks. Vol. 19. Chapter 19, 329–352.
[19] 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
[20] Cleotilde Gonzalez. 1996. Does animation in user interfaces improve decision making?. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 27–34. https://doi.org/10.1145/238386.238396
[21] Lars Grammel, Chris Bennett, Melanie Tory, and Margaret-Anne D Storey. 2013. A Survey of Visualization Construction User Interfaces.. In *EuroVis (Short Papers)*. https://doi.org/10.2312/PE.EuroVisShort.EuroVisShort2013.019-023
[22] John A Greenwood and Mark Edwards. 2009. The detection of multiple global directions: Capacity limits with spatially segregated and transparent-motion signals. *Journal of vision* 9, 1 (2009), 40–40. https://doi.org/10.1167/9.1.40
[23] Jeffrey Heer and George Robertson. 2007. Animated transitions in statistical data graphics. *IEEE Transactions Visualization and Computer Graphics* 13, 6 (2007), 1240–1247. https://doi.org/10.1109/TVCG.2007.70539
[24] HENRI. 2019. 40 Javascript UI Animation Libraries For Web & Mobile. https://bashooka.com/coding/40-javascript-ui-animation-libraries-for-web-mobile/. [Online; accessed 5-February-2020].
[25] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In

Proc. SIGCHI Conference on Human Factors in Computing Systems. 3363–3372. https://doi.org/10.1145/1978942.1979444

[26] Kurt Koffka. 1922. Perception: an introduction to the Gestalt-Theorie. *Psychological Bulletin* 19, 10 (1922), 531. https://doi.org/10.1037/h0072422

[27] 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 *Proc. SIGCHI Conference on Human Factors in Computing Systems*. 123. https://doi.org/10.1145/3173574.3173697

[28] Kiln Enterprises Ltd. 2020. Flourish. https://flourish.studio. [Online; accessed 5-February-2020].

[29] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. 2007. Show me: Automatic presentation for visual analysis. *IEEE Transactions Visualization and Computer Graphics* 13, 6 (2007), 1137–1144. https://doi.org/10.1109/TVCG.2007.70594

[30] Kim Marriott and Bernd Meyer. 1998. *Visual language theory*. Springer Science & Business Media.

[31] Stefan Mateeff, George Dimitrov, and Joachim Hohnsbein. 1995. Temporal thresholds and reaction time to changes in velocity of visual motion. *Vision research* 35, 3 (1995), 355–363. https://doi.org/10.1016/0042-6989(94)00130-e

[32] Gonzalo Gabriel Méndez, Miguel A Nacenta, and Sebastien Vandenheste. 2016. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 4073–4085.

[33] Microsoft. 2021. Windows Movie Maker. http://moviemaker.support/. [Online; accessed 5-September-2020].

[34] Tamara Munzner. 2014. *Visualization analysis and design*. CRC press.

[35] Thomas Lin Pedersen and David Robinson. [n.d.]. A Grammar of Animated Graphics | gganimate. https://gganimate.com. [Online; accessed 6-April-2020].

[36] Charles Perin. 2014. *Direct manipulation for information visualization*. Ph.D. Dissertation.

[37] Steven P Reiss. 1984. An approach to incremental compilation. *ACM SIGPlan Notices* 19, 6 (1984), 144–156. https://doi.org/10.1145/502949.502889

[38] Donghao Ren, Matthew Brehmer, Bongshin Lee, Tobias Höllerer, and Eun Kyoung Choe. 2017. ChartAccent: Annotation for data-driven storytelling. In *Proc. IEEE Pacific Visualization Symposium*. 230–239. https://doi.org/10.1109/PACIFICVIS.2017.8031599

[39] Donghao Ren, Bongshin Lee, and Matthew Brehmer. 2018. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions Visualization and Computer Graphics* 25, 1 (2018), 789–799. https://doi.org/10.1109/TVCG.2018.2865158

[40] Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche. 2018. Reflecting on the Evaluation of Visualization Authoring Systems: Position Paper. In *2018 IEEE Evaluation and Beyond-Methodological Approaches for Visualization (BELIV)*. IEEE, 86–92.

[41] 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

[42] Ray Rischpater and Daniel Zucker. 2010. Introducing Qt Quick. In *Beginning Nokia Apps Development*. Springer, 139–158.

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

[44] Hugo Romat, Caroline Appert, Benjamin Bach, Nathalie Henry-Riche, and Emmanuel Pietriga. 2018. Animated edge textures in node-link diagrams: A design space and initial evaluation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13. https://doi.org/10.1145/3173574.3173761

[45] Hans Rosling. 2006. Debunking myths about the "third world". TED (Technology, Entertainment, Design) Conference presentation. https://www.gapminder.org/videos/.

[46] Hans Rosling. 2007. The seemingly impossible is possible. TED (Technology, Entertainment, Design) Conference presentation. https://www.gapminder.org/videos/.

[47] James Rumbaugh, Ivar Jacobson, and Grady Booch. 1999. The unified modeling language. *Reference manual* (1999).

[48] 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

[49] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2019. Critical reflections on visualization authoring systems. *IEEE Transactions Visualization and Computer Graphics* 26, 1 (2019), 461–471. https://doi.org/10.1109/TVCG.2019.2934281

[50] Maruthappan Shanmugasundaram, Pourang Irani, and Carl Gutwin. 2007. Can smooth view transitions facilitate perceptual constancy in node-link diagrams?. In *Proceedings of Graphics Interface*. 71–78. https://doi.org/10.1145/1268517.1268531

[51] Ben Shneiderman. 1992. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics* 11, 1 (1992), 92–99. https://doi.org/10.1145/102377.115768

[52] Harmeet Singh and Mehul Bhatt. 2016. *Learning Web Development with React and Bootstrap*. Packt Publishing Ltd.

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

[54] TechSmith. 2020. Camtasia: Screen Recorder & Video Editor. https://www.techsmith.com/video-editor.html. [Online; accessed 15-September-2020].

[55] Tatiana Tekušová, Jörn Kohlhammer, Slawomir J Skwarek, and Galina V Paramei. 2008. Perception of direction changes in animated data visualization. In *Proceedings of the 5th symposium on Applied perception in graphics and visualization*. 205–205. https://doi.org/10.1145/1394281.1394331

[56] 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

[57] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. 2002. Animation: can it facilitate? *International journal of human-computer studies* 57, 4 (2002), 247–262. https://doi.org/10.1006/ijhc.2002.1017

[58] Yong Wang, Daniel Archambault, Carlos E Scheidegger, and Huamin Qu. 2017. A vector field design approach to animated transitions. *IEEE Transactions Visualization and Computer Graphics* 24, 9 (2017), 2487–2500. https://doi.org/10.1109/TVCG.2017.2750689

[59] Colin Ware. 2004. Information Visualization: Perception for Design.

[60] Colin Ware, Eric Neufeld, and Lyn Bartram. 1999. Visualizing causal relations. In *Proceedings of IEEE Information Visualization*, Vol. 99. 39–42.

[61] Daniel Weiskopf. 2004. On the role of color in the perception of motion in animated visualizations. In *IEEE Visualization 2004*. IEEE, 305–312. https://doi.org/10.1109/visual.2004.73

[62] 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

[63] Hadley Wickham. 2016. *ggplot2: elegant graphics for data analysis*. Springer.

[64] Haijun Xia, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor. 2018. DataInk: Direct and creative data-oriented drawing. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3173574.3173797

[65] Yingkui Yang, Hans Stubbe Solgaard, and Jingzheng Ren. 2018. Does positive framing matter? An investigation of how framing affects consumers' willingness to buy green electricity in Denmark. *Energy research & social science* 46 (2018), 40–47. https://doi.org/10.1016/j.erss.2018.06.006

[66] Jiayi Eris Zhang, Nicole Sultanum, Anastasia Bezerianos, and Fanny Chevalier. 2020. DataQuilt: Extracting Visual Elements from Images to Craft Pictorial Visualizations. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*. 1–13. https://doi.org/10.1145/3313831.3376172