Authoring Data-Driven Chart Animations through Direct Manipulation

Yuancheng Shen*, Yue Zhao*, Yunhai Wang, Tong Ge, Haoyan Shi, and Bongshin Lee

***Abstract*—We present an authoring tool, called CAST+ (Canis Studio Plus), that enables the interactive creation of chart animations through the direct manipulation of keyframes. 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 [1], 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. We report a user study conducted to assess the visual specification and system usability with its initial version. We enhanced the system's expressiveness and usability: CAST+ now supports the animation of multiple types of visual marks in the same keyframe group with new auto-completion algorithms based on generalized selection. This enables the creation of more expressive animations, while reducing the number of interactions needed to create comparable animations. We present a gallery of examples and four usage scenarios to demonstrate the expressiveness of CAST+. Finally, we discuss the limitations, comparison, and potentials of CAST+ as well as directions for future research.**

***Index Terms*—Chart animation, chart animation authoring, chart animation specification, data visualization, interactive system.**

I. Introduction

Chart animations are a powerful means for communicating data-driven insights: they are effective in attracting and retaining audiences' attention. Hans Rosling's animated bubble charts [2], [3] and numerous compelling examples [4], [5] 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 [6], Flourish [7], Adobe Stock [8]) 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, and 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 [9]) 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 [10] and gganimate [11] 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 [1], 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 timing factors.

In a previous paper [12], we presented CAST (**Ca**nis **St**udio), 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 the major visual components: keyframes. Combined 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. 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 keyframe synchronizing. After constructing keyframes and sequences with 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. The easy-to-understand visual specification of chart animations and this simple form of user interaction powered by auto-completion make CAST accessible to novices who cannot program chart animations.

Yuancheng Shen, Yue Zhao, Tong Ge, and Haoyan Shi are with Shandong University, China. Email: {remoteshen, jack.zhao9802, tgeconf, crmocreation}@gmail.com.

Yunhai Wang is with Renmin University of China. E-mail: wang.yh@ruc.edu.cn

Bongshin Lee is with Yonei University, Seoul, Republic of Korea. E-mail: b.lee@yonsei.ac.kr.

* The authors contributed equally to this research. Yunhai Wang and Bongshin Lee are co-corresponding authors.

Manuscript received October 10, 2023; revised August 10, 2024; accepted October 28, 2024.

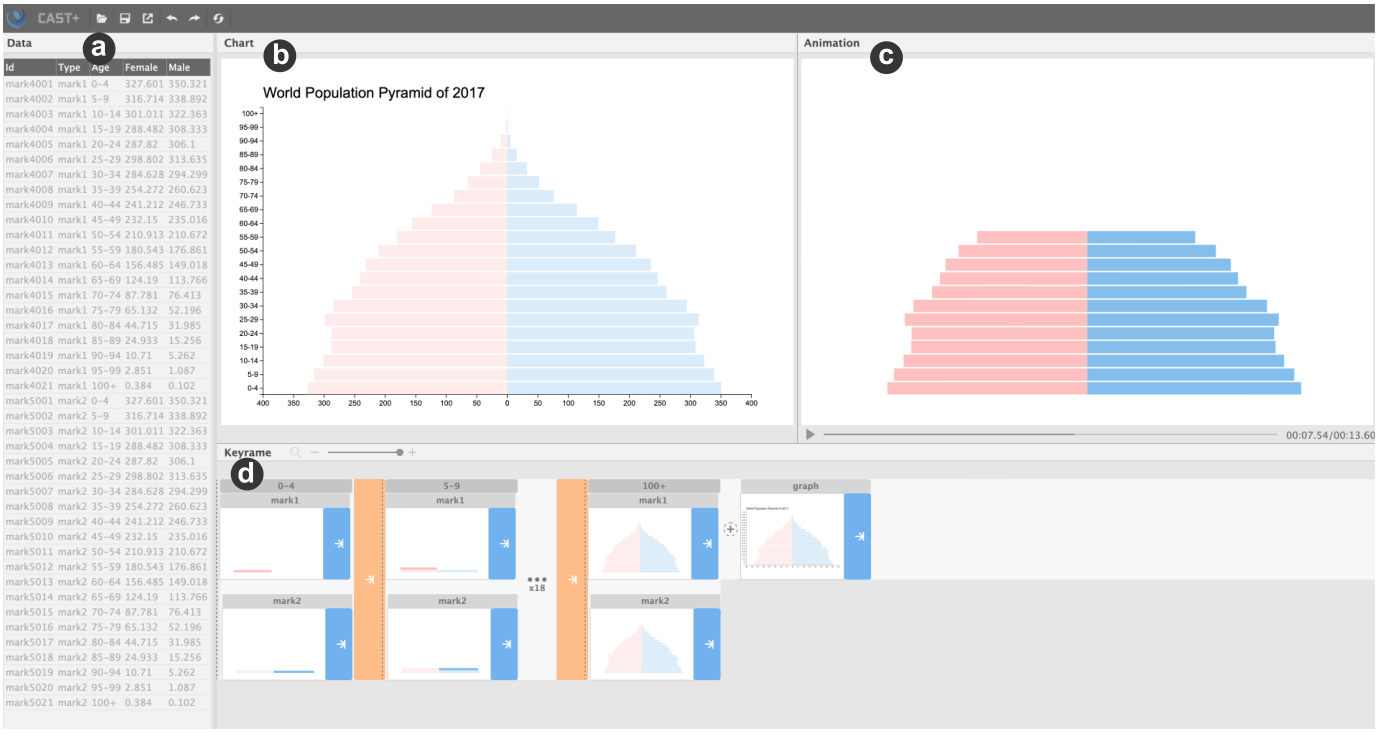


Fig. 1: CAST+ enables the interactive construction of a variety of data-driven chart animations. Its interface consists of four panels: (a) data panel; (b) chart panel; (c) animation panel; and (d) keyframe panel. In this case, the system is about halfway through the animation of a *'diverging bar chart,'* showing the distribution of the population by gender in different age groups. Please visit the CAST+ website (https://canisstudio.github.io/CASTPlus) to see the animation.

This paper presents CAST+ (**Ca**nis **St**udio Plus; Figure 1) that enhances the old system's expressiveness and enables the easier creation of animations featuring diverse visual marks within a keyframe group. We accomplish this by extending the Canis grammar and enhancing the auto-completion algorithms based on generalized selection [13]. This results in fewer interactions to achieve comparable animations and allows for the production of more expressive ones. In addition, we refined the user interface and interaction to provide more consistent interactions for both individual keyframes and keyframe groups and to offer data-driven timing specifications.

In summary, the main contributions of this work are:

- We introduce a visual specification for chart animations that explicitly represents keyframe-based animations, facilitating the direct manipulation of keyframes for authoring chart animations.
- Based on our visual specification, we design and develop CAST+, a web-based system that enables the interactive construction of chart animations. CAST+ enhances auto-completion by employing generalized selection to reduce the effort required in constructing keyframes and keyframe sequences, and direct manipulation to facilitate the easy specification of chart animations.
- We present three 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 using CAST, the initial version of the system. A comparison of keyframe specification interactions between CAST and CAST+ demonstrates the improvements of CAST+ in reducing low-level interactions and minimizing users' cognitive load.

II. 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 [1]. In this section, we focus on the creation of chart animations and the reader can refer to previous studies [14]–[16] for the effectiveness of chart animations.

A. Perception of Animations

A complete review of general perception of animations is beyond the scope of this paper (refer to Tversky et al. [17] and Chevalier et al. [18]). We restrict our discussion to the perception of data-driven chart animations, where motion is used as a preattentive visualization technique [19]. Previous studies show that appropriately-designed animations not only reveal complex data relations (e.g., causal relations [20]) but also support filtering and brushing [21] as well as facilitate decision making [22].

Following the Gestalt law of common fate [23], 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 [24], direction [25], or

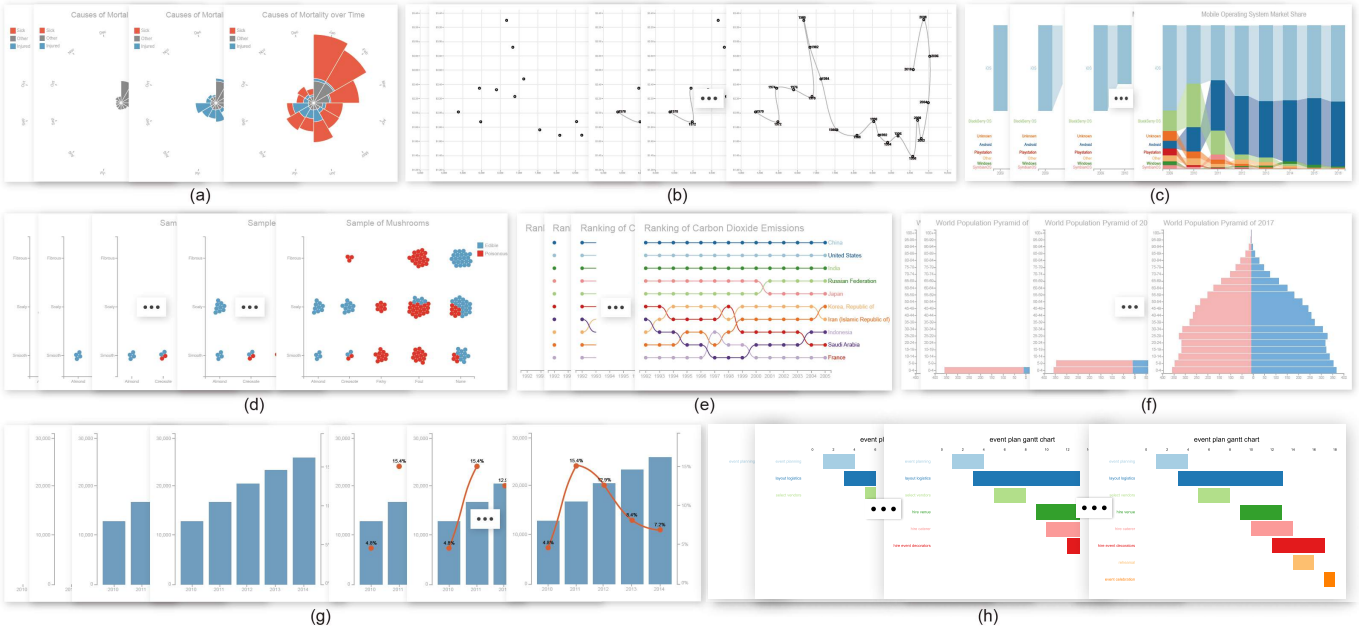


Fig. 2: Eight example chart animations demonstrating the expressiveness of CAST+, where the ones in (a-g) can be created by both CAST and CAST+. We used the first seven chart animations (a-g) as tasks in our user study (Section IV). More examples can be found with the input charts, descriptions, and videos illustrating the creation processes in the website (https://canisstudio.github.io/CASTPlus).

both [26] in animated visualization. Weiskopf [27] found that color plays an important role in the perception of motion direction and velocity. Romat et al. [28] introduced three motion variables (speed, frequency, and pattern) for defining animated edge textures in node-link diagrams and assessed their effectiveness. Recently, Chalbi [29] suggested that not only the motion but also the change in luminance and size all have a strong grouping effect. Although CAST+ does not fully guarantee the perceptual effectiveness of chart animations, 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.

B. Chart Animation Design Space

Animation is a promising way for conveying the changes in visualizations, although its effectiveness is still controversial for data analysis [17]. To promote effective animation design, a few guidelines have been proposed. Tversky et al. [17] 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 [13] and Fisher [30] 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. [4] characterize the elemental units of data videos as the combinations of visualization types $\times$ animation types. More recently, Thompson et al. [5] 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. These dimensions can be combined into compositions of animated transitions and pacing techniques. While not perfectly aligned with Thompson et al.'s design space, CAST+'s design can be projected onto their space. CAST+ enables users to select *graphic objects*, such as marks, (title) text, axis, and 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 (e.g., staging, staggering, and trajectories) have also been studied. Bartram and Ware [21] find that objects with similar motions are preattentively grouped even if they are otherwise dissimilar. Heer and Robertson [13] show that carefully designed staged animation improves understanding of the underlying data, while complex multi-stage transitions are less favored. Shanmugasundaram et al. [31] find that smooth animation transitions help in maintaining the connectivity and overall structure in node-link diagrams. Dragicevic et al. [32] 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 animation transition types. Chevalier et al. [33] study another pacing technique, staggering, where the start time of moving elements is delayed incrementally and find that it has a negligible, or even negative impact on multiple objects tracking performance. Other studies find that bundled trajectories or smooth non-linear ones [34], [35] are more effective than straight ones in some conditions. CAST+ is designed to help users easily

follow these guidelines with a simple and intuitive interface.

C. Chart Animation Authoring Tool

A survey of visualization authoring tools is referred to Grammel et al. [36] 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 [37], [38] have been developed for creating general animations, but the ones specifically for chart animations are scarce. One striking example is D3 [10], 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 [39] 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 resulting verbose specification impedes rapid authoring.

In contrast, gganimate [40] provides a high-level grammar for easily creating chart animations. However, it can only animate the charts created by ggplot2 [41] with limited types of animation transitions. By separating the chart animation from the creation step, the recently proposed high-level chart animation language, Canis [1], enables the concise, high-level specification of chart animations for the input of any data-enriched SVG (dSVG) chart. Likewise, Gemini [42] provides a declarative grammar for specifying chart animations but focuses on creating staged animations between keyframes. Gemini^2 [43] automatically recommends intermediate keyframes, which significantly simplifies the animation specification process. Animated Vega-Lite [44], an extension of Vega-Lite [45], provides a unified abstraction for static, interactive, and animated visualizations. By modeling animated visualizations as time-varying data queries, authors can seamlessly specify or move between static, interactive, and animated visualizations. However, these tools still 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. [5], 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 [6] allows non-experts to craft data videos by composing a sequence of predefined combinations of visualization types and animation types. Likewise, Adobe Stock [8] provides a set of data-driven motion graphics templates, while Flourish [7] 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, keyframe-based tools, like Adobe After Effects [9], 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.

The most closely related work is Data Animator [46], designed for authoring keyframe-based animations through direct manipulation. However, it is limited to input charts created by Data Illustrator [47] and focuses on specifying animated transitions between keyframes rather than animations of a chart. Besides, it has three major differences from CAST+ in keyframe manipulation: (1) it uses an abstract representation (*i.e.*, a circle) of a keyframe on the timeline, while CAST+ has an explicit representation; (2) it requires users to pre-define all keyframes for generating automated transitions between two adjacent ones; and (3) it does not support semantic zooming, essential for displaying numerous keyframes at different levels of detail. In contrast, CAST+ allows for visually specifying animations to the charts created by any visualization tools and provides data-driven auto-completion based on generalized selection [48] for authoring expressive chart animations and semantic zooming for exploring numerous keyframes.

D. Visual Specifications

The visual specification approach describes an abstract description of an object or a system by using a graphical vocabulary. Thus, it makes specification easier and more accessible with no need to know the details of how the system works [49]. The most notable example is Unified Modelling Language (UML) [50], which provides a variety of visual objects for modeling software systems.

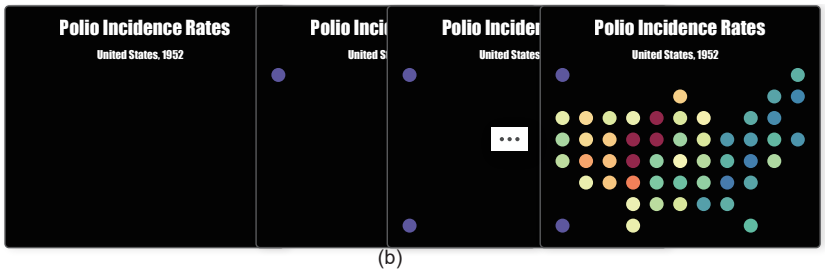
Many visualization systems offer interactive visual specifications of visualization and analysis operations. Polaris [51] and its commercial successor Tableau [52] allow users to define visual specifications via drag-and-drop operations, namely, placing data fields onto "shelves" corresponding to visual encodings such as position, size, shape, or color. iVoLVER [53] provides an interactive visual language that enables users to acquire data from various sources and create interactive visualizations and animations. Lyra [54] allows visual specification of flexible custom visualizations such as force-directed layouts, trees and word clouds. Wrangler [55] 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.

E. Canis: CAST+'s Underlying Grammar

CAST+ is built on *Canis* [1], 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

```
"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)

Fig. 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.

(https://chartanimation.github.io/canis) for the complete specifications and example animations. Later in Section III-F, 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 keyframes defined by animation units *aniunits*. The input charts are embedded with the source data, and each *aniunit* is defined by a quadruple:

$$aniunit := (selector, timing, grouping, effects), \quad (1)$$

where the *selector* is used to select marks to be animated from the input charts using the W3C Selectors API [56], and timing, grouping, and effect are all properties of a keyframe. The marks selected via the *selector* operator can be divided into a set of elementary units, referred to as *mark units*, by *grouping* them with categorical or nominal data attributes. 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 the 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).

Figure 3 (a) shows an example of Canis specification. It consists of two aniunits that 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).

III. CAST+

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

A. Design Principles

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

DP1: Provide a literal representation for keyframes in a chart animation. Existing keyframe-based systems represent animations in an abstract manner in 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). Like After Effects, Data Animator [5] represents a keyframe using abstract/symbolic diamonds and circles on the timeline.

DP2: Use grouping and semantic zooming to support keyframe and keyframe sequence reading. While a literal representation provides many benefits for end-users, it poses challenges when dealing with a large number of marks and keyframes. To mitigate this issue, we support keyframe grouping, which allows for nesting and grouping at each level based on a unique visually encoded data attribute. Additionally, we introduce semantic zooming [57] in the animation specification panel to ensure the visibility of visual marks while preserving the hierarchical structures.

DP3: Use grouping and generalized selection to support keyframe and keyframe sequence authoring. The straightforward method for constructing a keyframe involves manually selecting all the necessary visual marks from the (dSVG) chart. However, this fully manual process is time-consuming, prone to errors, and becomes impractical as the number of keyframes increases. CAST+ aims to balance flexible graphics manipulation with procedural keyframe sequence generation based on Canis. It enables efficient keyframe construction through generalized selection [48], which suggests visual marks based on the data attributes of the user-selected marks. Additionally, CAST+ recommends specific and meaningful keyframe sequences based on the constructed keyframes through speculative generalized selection.

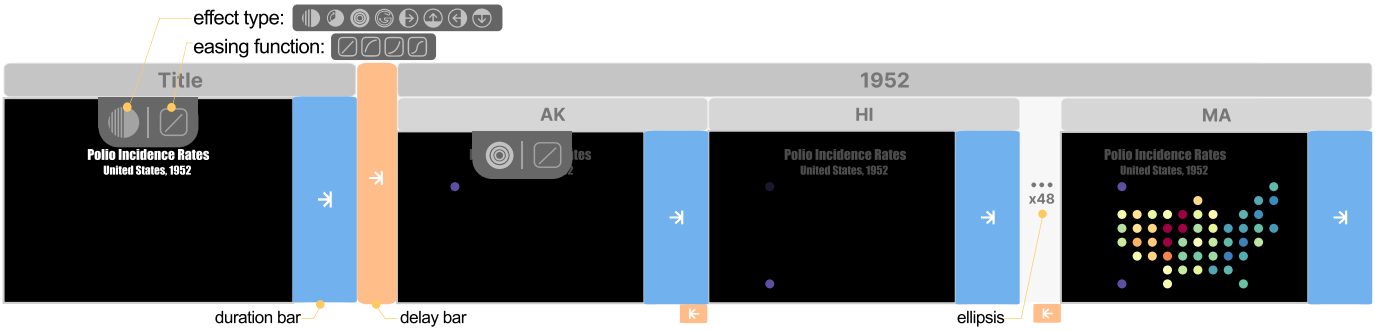


Fig. 4: The visual representation of the animation of a map showing the polio incidence rates of the United States in 1952. The title fades in first, then all dots animate according to their rate in an ascending order with the "circle" effect. Such animation is represented by two keyframe groups with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

B. 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 II-E, a mark unit refers to a collection of marks manually or automatically selected with the underlying 'selector' operator in Canis, 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 based on different visually encoded data attributes, and the level of each group is indicated using nested boxes with different shading intensities (mimicking the treemap [58]). These keyframes and keyframe groups convey meaningful steps in the animation, forming a storyboard. Each keyframe and keyframe group (in CAST+) has a header showing the chart component (e.g., Title) or data value (e.g., 1952, AK, HI, MA) that characterizes the animation to facilitate the understanding of the hierarchical grouping structures. For example, as shown in Figure 4, the animation of a scatterplot incrementally revealing the polio incidence rates of the United States in 1952 can be represented with two keyframe groups: the 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 state 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)[1].

To map the 'timing' property of keyframes in Canis, 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. On the other hand, each keyframe group (in CAST+) can have its own delay bar (e.g., see the tall orange bar in Figure 4).

CAST+ allows a keyframe to start before the previous keyframe finishes. In such case, the delay bar of the next keyframe and the duration bar of the previous keyframe will be right-aligned, indicating a *negative* delay. For example, in Figure 4, the delay bar of the second keyframe (HI) is right-aligned with the previous keyframe (AK), conveying that the animation for the second keyframe will start while the first keyframe is still animating. To avoid complete occlusion between them, the delay bar is scaled in the Y direction to be a bit taller than the duration bar.

All keyframes in the same keyframe group share animation properties, and thus showing all keyframes in a keyframe group is redundant, consuming large screen space. Therefore, CAST+ shows only three (first, second, and last)[2] 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. CAST+ maps two properties of animation effects in Canis—effect type and easing function—by using icons. It 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.

C. User Interface and Interaction

The CAST+ user interface consists of a data panel, a chart panel, an animation preview panel, and an animation

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

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

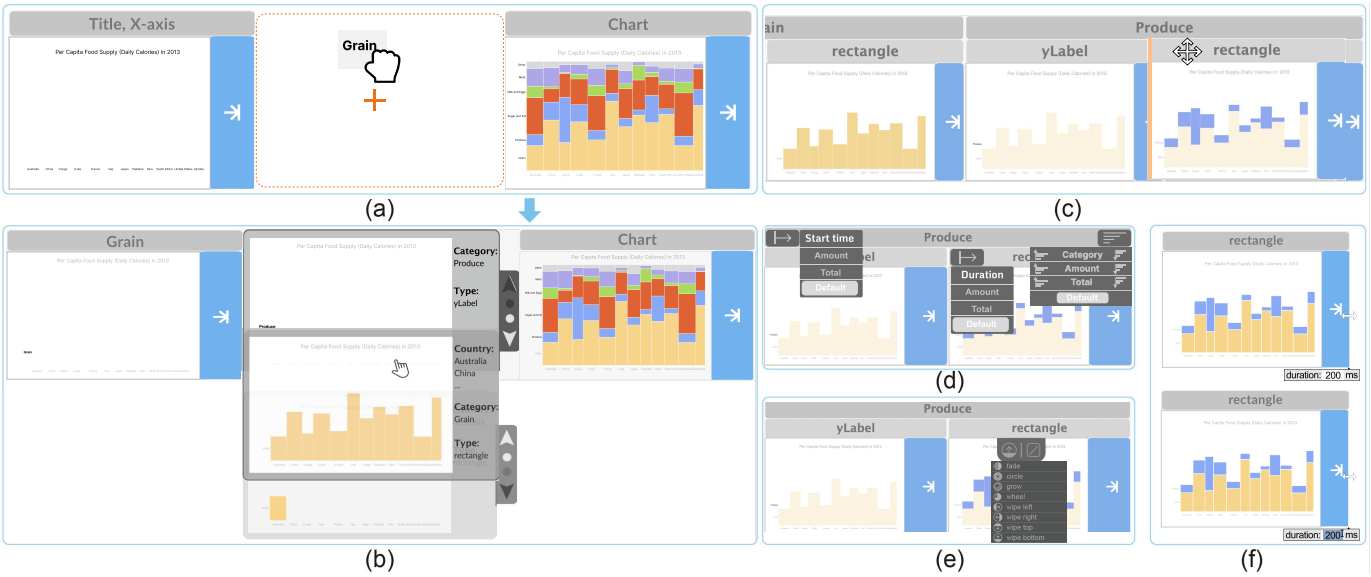


Fig. 5: An example procedure of the keyframe and keyframe sequence construction. (a) a tick label is dragged over the dropzone ahead of the keyframe group containing visual marks; (b) after creating a keyframe for the selected label, a list of possible next keyframes is suggested to let the author select to complete the keyframe sequence; (c) dragging one keyframe to make keyframes animate a short time after the previous one started; (d) binding quantitative data attributes to the start time or duration of the keyframes and rearranging the animation order of sibling keyframe groups; (e) selecting effect type for keyframe group; and (f) two ways to update the duration length.

specification panel (Figure 1). We explain CAST+'s interaction using five 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://canisstudio.github.io/CASTPlus).

Interaction Mechanisms. When the data-enriched SVG chart is loaded[3], 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 create the customized animation with the chart, There are three ways to specify keyframes: constructing keyframes, selecting the next keyframe, and synchronizing keyframes. Hence, the author starts the process of creating the mark unit to be animated by 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 displayed (for example, see Scenario 2). Moreover, when it is difficult to select marks in a few selection operations, (e.g., selecting all blue 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 suggested by CAST+ through the extended generalized selection [48] or keep selecting from the input chart manually.

Once the mark unit is selected, the author can drag and drop them on the animation specification panel to generate a keyframe (see Figure 5 (a)). Then, the system suggests a list of potential next keyframes for the author to select (see the center in Figure 5 (b)). To facilitate the selection of the desired next keyframe, these suggested keyframes and the chart panel are linked together (see video in the supplemental material). Specifically, the mark units associated with the potential next keyframes are highlighted in the chart panel. When the author hovers over each suggested keyframe, the corresponding visual marks are highlighted in the chart panel. When a suggested keyframe is chosen, the system continues to suggest the next keyframe until the keyframe sequence is finalized In doing so, the 'selector' concept in Canis is mapped to the suggestion-selection interaction for creating keyframes and keyframe sequences in CAST+.

Once the keyframe sequence is generated, the author can synchronize keyframe groups by rearranging the animation order of the sibling keyframe groups with the popup panel on their parent keyframe group (Figure 5 (d)). To synchronize keyframes, the author can drag the header of one keyframe to a new location and adjust its relative position to the previous one. CAST+ hints at the changes to be made while the keyframe is being dragged (Figure 5 (c)). The author can change the timing of keyframe groups in the same way. However, CAST had an interaction inconsistency: the author drags a bar on the bottom of the keyframe but the tab that appears on the top left corner of one keyframe group on mouse hover, see the illustration in the supplemental material.

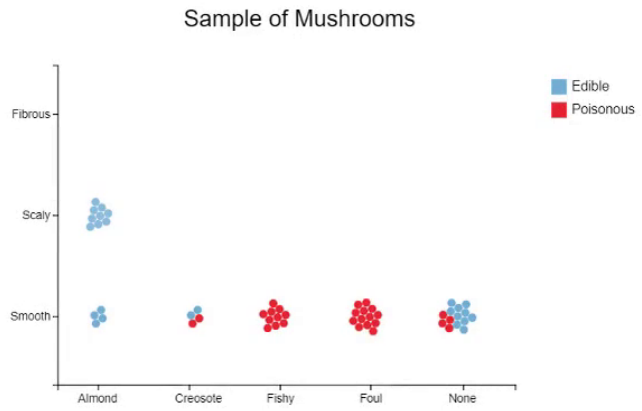
As for editing the length of duration or delay, the author can change the width of the timing bar by dragging the end of it (Figure 5 (f)). To specify precise duration or delay,

[3]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.

they can enter the number in the popup input box shown when hovering the timing bar (Figure 5 (f)). As CAST+ now supports data-driven timing, the author can bind a quantitative attribute to the start time or duration in CAST+ (Figure 5 (d)). Likewise, she can also sort the keyframes or keyframe groups in terms of a quantitative data attribute (Figure 5 (d)). In doing so, the authored data-driven animations can more effectively reflect data patterns. 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, enabling 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.)

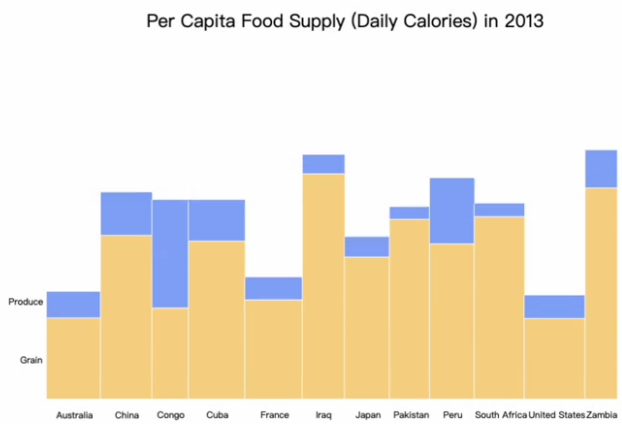
Scenario 1: Animation of the Faceted Dot Plot.



We will create the animation with a faceted dot plot, 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 the animation of cells within the same row and a longer pause between rows.

To create this animation, we first animate the title, axis, and legend, which 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 in the bottom left corner. Then the system provides a list of the next keyframes indicating all possible keyframe sequences, where each one corresponds to the unique partition strategy for all dots. We choose the cell with two red and two blue dots to the right of the four selected blue dots in the same row, indicating that the attribute `isEdible` does not influence the construction of keyframe sequences. As a result, we obtain the desired keyframe sequence, labeled to partition first by `Surface`, then by `Odor`. Finally, to add delay between keyframes, we adjust the distance between two adjacent keyframes by dragging one away from the other. After inserting a delay between the keyframe groups in the same manner, we increase the delay by stretching the right border of the delay bar. Note that the user interaction is almost the same for creating this animation with CAST and CAST+.

Scenario 2: Animation of the Mekko Chart.



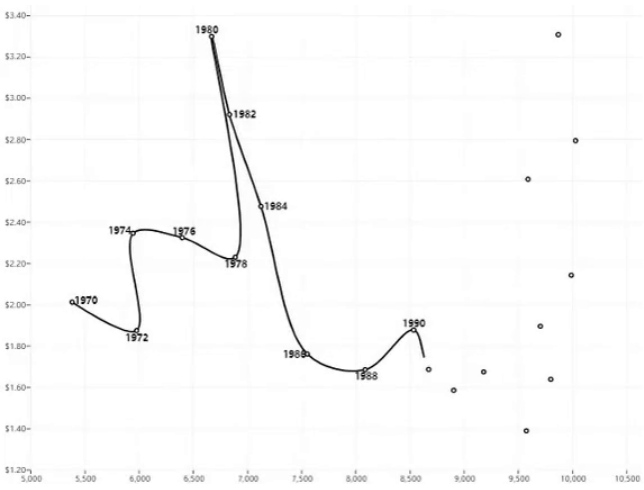
We will create an accumulation animation with a Mekko chart, which shows per capita food supply. In the chart, each column represents the proportion of calories provided by different foods in one country, and the color of each rectangle encodes the type of food. The animation starts by fading in the 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 the title and axes, we drag them from the chart panel to the animation specification panel to create the first keyframe. Yet, implementing the rest of the animations is challenging with CAST. Since it can only perform the auto-completion for the marks shared with the same data attributes, we have to separately generate the animations of the rectangles and tick labels. We animate rectangles by food type by first selecting the rectangles corresponding to `Grain` to create another keyframe. In response, the system 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, which is a tedious and time-consuming process.

In contrast, creating this animation using CAST+ is straightforward. After creating the keyframe with the title and axes, we drag the label at the bottom row to the animation specification panel, and then CAST+ subsequently suggests the next keyframes including the bars in the same row and the label in the next row. We select the bars to form a keyframe group due to the underlying nested selection in Canis, which includes the y-axis label and all corresponding rectangles (see more detail in Figure 8), subsequently generating the entire keyframe sequence. Due to the keyframe group, we do not need

to synchronize the keyframes. Changing the animation effect of the bars to "wipe bottom" achieves the desired animation.

Scenario 3: Animation of the Connected Scatterplot.

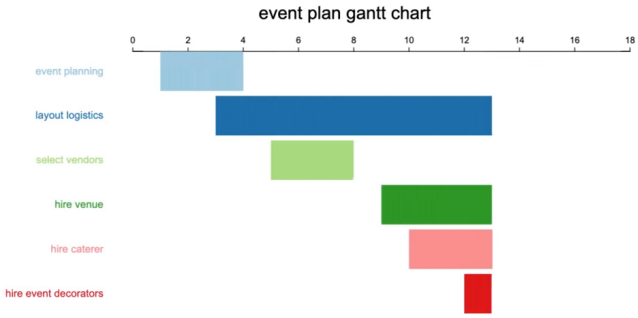


We will create an animation with Hannah Fairfield's connected scatterplot [59], which shows the relationship between driving distance and oil price over time. In this chart, each dot representing the oil data of a year is positioned according to miles driven (x-axis) and its price (y-axis), having year 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 with CAST, we select several axis ticks with their labels from the input chart. Then the system 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 the dots in a similar way. In response, the system 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.

With CAST, crafting this animation requires users to manually synchronize different keyframe groups for simultaneously animating labels and links, which is a time-consuming process. In contrast, this can be easily achieved with CAST+. After selecting axis ticks, axis labels, grids, and all dots as the starting keyframes, we simply need to simultaneously select the link and year label, with the support of nested selection in Canis, as a keyframe and drag them to the animation specification panel. CAST+ then auto-completes all the subsequent keyframes, given that only one possible sequence remains, without the need of synchronizing keyframes. Finally, we change the animation properties of the links as we do in CAST.

Scenario 4: Animation of the Gantt Chart.



We will create an animation with a Gantt chart. This chart visualizes a project schedule, where each horizontal bar represents a task with a label (task name) having a unique color. Each task bar is positioned based on when the task is scheduled to start, with its length encoding the task duration. The animation of this chart starts by fading in the title and axis, followed by the gradual unveiling of tasks' timelines one by one. By mapping the total project duration to the animation duration, the label and the bar of each task appear at the project starting time with fade-in and wipe-left effects, respectively. The duration of the animation on each bar is proportional to the corresponding task's duration.

It was not possible to create this animation with CAST because the timing bar can only specify a constant duration and delay to the same keyframe group. To address this limitation, CAST+ allows for binding quantitative data attributes to animation timing properties. The animation creation process begins by selecting the title and axis and dragging them to the animation specification panel to create the first keyframe. Then we select the label of the first task and drag it to the animation specification panel. Subsequently, CAST+ suggests the next keyframe list consisting of the bar of the first task and the label of the second task. Here, we select the bar to generate a keyframe group, including label and all related bars together, and then CAST+ auto-completes the entire animation. To consistently reflect the schedule, we bind the group's start time with the `Begin` data attribute and map the bar's duration with the `Total` data attribute. Then we modify the animation effect of the bars to "wipe left". Finally, we drag the keyframe group of bars to align with one of the labels for animating them simultaneously.

D. Auto-completion of Keyframes and Keyframe Sequences

In this section, we explain how CAST+ supports auto-completion through generalized selection for constructing both keyframes and keyframe sequences (**DP3**). This is achieved through two major components: mark-type-aware keyframe auto-completion and bottom-up next keyframe recommendation. For each component, we first describe how CAST operates and then compare it with the CAST+ approach.

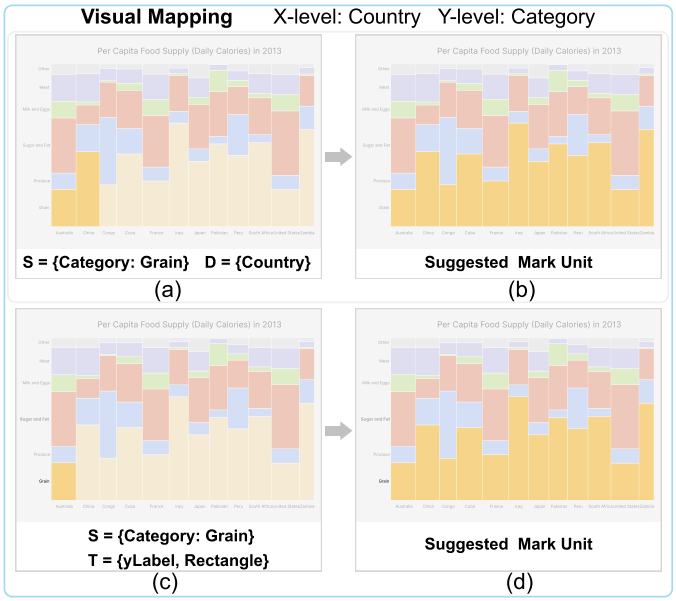


Fig. 6: Unlike CAST, CAST+ can suggest additional marks when *multiple* types of marks are selected by the user. When two rectangles in yellow from different countries are selected (a), all yellow rectangles are highlighted as a suggestion result (b). When a label in the y-axis and its corresponding rectangle of the first country are selected (c), the system suggests all other yellow rectangles (d).

Mark-type-aware Keyframes Auto-completion. Given the input chart, CAST infers the underlying mark units based on the selected visual marks and recommends all marks in the corresponding unit for completing the selection. Assuming that users at least made two selections with the set of marks $\mathbf{M} = \{M_1, M_2, \cdots, M_n\} (n \geq 2)$, CAST infers the mark unit by comparing the visually encoded data attributes. Specifically, CAST constructs two sets: S and D where each item in S is an attribute $attr$ and the data value val shared by all marks and each item in D is an attribute with different data values in the marks M_i and M_{i+1}. For example, S only has the attribute *Category* with the value `Grain` in Figure 6(a), while D includes `Country`.

Accordingly, the mark unit consists of the visual marks having the same value for each attribute in S and all different values of each attribute in D, and all un-selected marks are suggested. In Figure 6(b), all yellow marks are highlighted and suggested for selection while the others are translucent. If multiple visual marks share the same data values for all attributes in S+D, CAST will randomly select one mark when M_i has only one mark. It may prioritize attributes with the most effective channels, following the visual encoding effectiveness principle [60].

By embedding the type of visual marks as an additional data attribute in the input data-enriched SVG file, we can use the previous algorithm to complete the keyframe construction once at least two types of visual marks are selected. Treating mark type as an attribute in D lets the system suggest different types of visual marks having the same value of the attributes in S for selection. In doing so, after selecting the dot and label of the year 1970 from the connected scatterplot in Scenario 3, the corresponding link will be suggested to be animated together. However, this algorithm requires that all selected marks are associated with the same data attributes, which limits its effectiveness. For example, if a user selects the axis label and rectangle in the Mekko example (Figure 6 (c)), the algorithm fails since the axis label possesses only the `Category` attribute and lacks others like `Country` associated with rectangles.

To address this issue, we propose a mark-type-aware auto-completion algorithm in CAST+, which suggests various types of visual marks for constructing keyframes. Inspired by generalized selection [48], CAST+ generalizes the selection to include additional related marks (*e.g.*, "select all marks like these ones"). Specifically, it examines the encoded data attributes of the selected visual marks to construct two sets S and T, where each item in S is an attribute and the data value shared by all selected marks and T consists of the unique type names of the selected marks. If S is not empty, all un-selected marks will be suggested if they meet two conditions: i) encoding the same data value of each attribute in S and ii) mark types belonging to one of these in T. This query process is described in Algorithm 1. Hence, the mark unit consists of visual marks with the same value for each attribute in S, and their mark type should match that of the selected marks. In doing so, the shared attribute of the selected label and rectangle in Figure 6 (c) is `Category`, resulting in a keyframe group that includes the label and all suggested yellow rectangles in Figure 6 (d).

Algorithm 1 The auto-completion algorithm of CAST+

Input: dSVG C, and two attribute sets S and T
Output: The suggested mark unit MU

```
function FindMarkUnit( C, S, T )
    MU = ∅
    for each mark ∈ C do
        if ⋀_{(attr,val)∈S} (mark.attr == val) ∧ mark.type ∈ T then
            MU = MU ∪ mark
        end if
    end for
    return MU
end function
```

Bottom-up Next Keyframe Recommendation. After constructing the first keyframe, the system suggests a list of unique next keyframes for the author to select. Assuming that each keyframe group consists of the same type of visual marks, CAST takes a top-down approach that first generates all potential animation keyframe sequences based on the permutations of all visually encoded data attributes and then iteratively filters data attributes. However, this assumption does not hold, when a mark unit has multiple mark types with different data attributes. For example, the selected y-axis label and the yellow bar in Figure 5(b) share only one data attribute and the next keyframes cannot be recommended.

To address the above issue, we introduce a bottom-up approach for recommending the next keyframes. Following the Gestalt principle of Common Fate [61], the visual marks in each

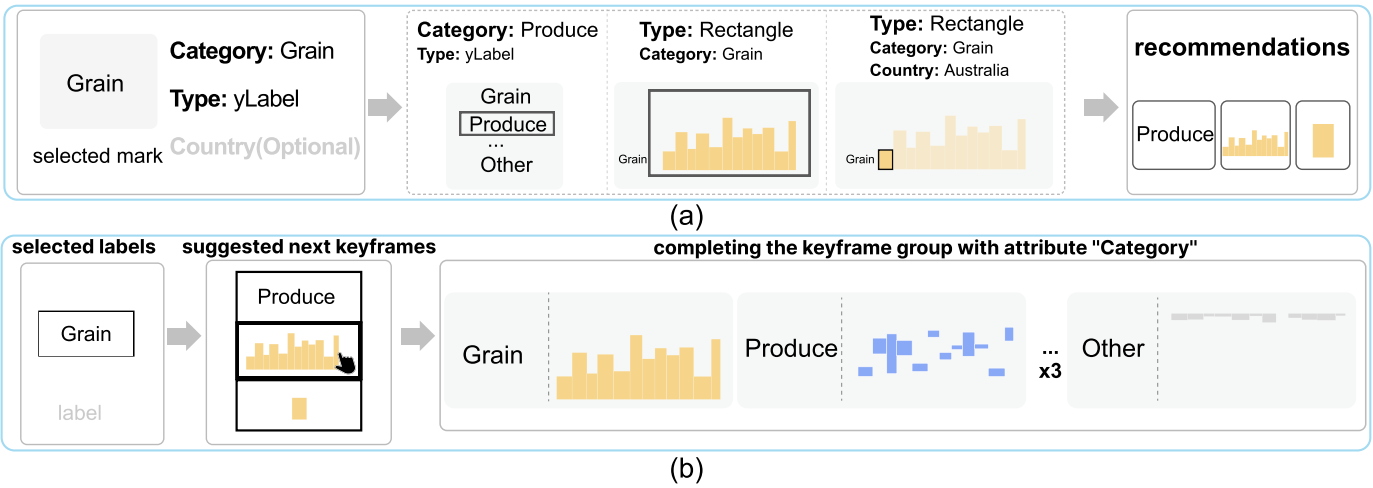


Fig. 7: The keyframe sequence auto-completion process with CAST+ includes the following steps: (a) Computing suggested next keyframes involves analyzing the data attributes associated with the selected label (left). This is followed by identifying three different keyframes (middle) by varying the value of one attribute or mark type, and finally, removing duplicates to form the suggested next keyframes (right). (b) Constructing the animation sequence begins with selecting a label (left). Next, a keyframe featuring all yellow bars is chosen from the list of suggested next keyframes (middle). Finally, the entire keyframe sequence is generated (right) by repeatedly changing the value of the "category" attribute.

keyframe group undergo similar visual changes and we believe that only one visually-encoded data attribute or mark type undergoes changes within a keyframe group helping viewers to understand. Given the selected mark unit in the current keyframe, we generate the recommendations of next keyframes using a speculative generalized selection technique. Rather than relying on a user's query to contain all the necessary information, CAST+ guesses possible query relaxations by considering every possible relaxation of a single attribute in the selection. We take the following steps to generate the next keyframes recommendations (as also outlined in Algorithm 2).

1) We first find the set of data attributes S that are commonly visually encoded by the selected visual marks in each of the provided keyframes (if there are more than one). The data attributes that are not encoded by the selected visual marks are placed into the set of optional attributes O. For example, the attribute `Country` is an optional one in Figure 7 (a). In addition, we place the mark type of each selected keyframe into a set T, and the mark types included in the given chart but not in the selected keyframes into a set nT.
2) We then identify potential mark units that differ from the current keyframe's unit by either one attribute value or by mark type.
   - If an attribute in S differs, we find a mark unit with this attribute having all possible values, while other attributes match the selected marks. Once a mark unit is found, we select the mark or marks nearest to the selected visual marks in the current keyframe, see the example of the label "Produce" in the middle of Figure 7 (b).
   - If the mark type differs, we consider the mark unit with all possible combinations of visual mark types. Hence, we construct a power set $\mathcal{P}(\mathrm{nT})$ for the set nT which includes all the corresponding subsets. For each member in $\mathcal{P}(\mathrm{nT})$, we find the mark unit with the attributes matching to the set S. For example, in Figure 7 (a), when users select the `Grain` label, three sets are generated T = {Label}, nT = {Rectangle}, $\mathcal{P}(\mathrm{nT})$ = {{}, Rectangle}. The mark unit with all yellow bars in the middle is then suggested. Since all marks in the selected mark unit might further encode some optional attributes, we further partition them by using the additional optional attributes defined in the power set $\mathcal{P}(\mathrm{O})$ of the set O and then find the partitioned mark unit nearest to the selected marks in the current keyframe. In the middle of Figure 7(a), the single yellow bar is generated by using the optional attribute `Country` to partition the mark unit containing all yellow bars and finding the nearest one to the selected label. Note that if only one suggested mark unit is found, we will continue to search for the next keyframe(s) until no more can be suggested. For example, in Figure 7(b), if users select the corresponding rectangles, only one suggestion with type T = {Label} will be provided. This process continues until the final gray rectangles are generated, after which no further suggestions are available. To learn how this algorithm works with various types of visual marks and multiple optional attributes, we refer to the supplementary material.
3) Finally, we eliminate empty sets and duplicate mark units, and rank them by measuring the transition visualization costs between the suggested mark unit and the one in the current keyframe [62].

Once the user selects one of the suggested next keyframes, the system automatically generates the next keyframes until no further suggestions are available. If there are still visual

Algorithm 2 Next Keyframes Recommendation

Input: dSVG C, the selected keyframes K
Output: Ranked next keyframes

function FindNextKfsOrKfGroup(C, K)
  [S,O] = FindCommonOptionalAttr(C,K) ▷ identify shared and optional attributes
  [T,nT] = FindMarkType(C,K) ▷ get present and absent mark types
  G = ∅
  $\mathcal{P}$(nT) = $\{S \mid S \subseteq \text{nT}\}$ ▷ power set of set nT
  $\mathcal{P}$(O) = $\{S \mid S \subseteq \text{O}\}$
  for each attr ∈ S **do** ▷ remove one data attribute
    restAttr = $\{(a, v) \in S \mid a \neq \text{attr}\}$
    G.push(FindNextKfs(C, K, restAttr, $\mathcal{P}$(O),T, False)) ▷ find keyframes with remaining attrs
  end for
  for each Ts ∈ $\mathcal{P}$(nT) **do** ▷ change the mark type
    G.push(FindNextKfs(C, K, S, $\mathcal{P}$(O), Ts, True))
  end for
  if |G| == 0 **then**
    return ∅
  else if |G| == 1 **then**
    G.push(FindNextKfsOrKfGroup(C,K∪ G))
    return G
  else
    result = RemoveDuplicates(G)
    return Rank(result) ▷ sort by transition cost
  end if
end function
function FindNextKfs(C, K, S, $\mathcal{P}(O)$, Types, isOptional)
  nextKfs =∅
  mUs = FindMarkUnit(C, S, Types) ▷ See Alg. 1
  if isOptional==False **then**
    nextKfs.push(GetNearestUnit(mUs)) ▷ get the closest matching marks
  else
    nextKfs.push(mUs)
    for each attr ∈ $\mathcal{P}(O)$ **do**
      mUs = PartitionUnit(mUs, attr) ▷ partition mUs based on optional attributes
      nextKfs.push(GetNearestUnit(K, mUs))
    end for
  end if
  return nextKfs
end function

marks that have not been animated, the user can select them to construct a keyframe and then the system will continue to recommend the next keyframes.

E. Keyframe Grouping and Semantic Zooming

In this section, we explain how CAST+ supports exploring numerous keyframes and keyframe sequences (DP2), and compare it with the approach used in CAST.

Keyframe Grouping. During the next keyframe selection, CAST automatically groups adjacent keyframes together by checking if there is only one common associated attribute having different values. For example, the dot cells within each row form a keyframe group in Scenario 1, which can be further hierarchically grouped.

Yet, each keyframe group in CAST consists of a set of the same type of visual marks due to the original Canis [1]. As shown in Figure 8 (a,b), the selected visual marks that encode the same data attributes form a mark unit tree. However, interlacing these two groups to achieve the animation described in Scenario 2 requires carefully adjusting the duration and delay for each keyframe and keyframe group as shown in Figure 8 (c). On the other hand, these two types of visual marks can be selected as one animation group by *"selector": ".y-axis-label, .rectangle"*. Yet, they only share one data attribute `Category` and thus can form a two-level mark unit tree, limiting its expressiveness.

To overcome this limitation, we extend the Canis grammar with a nested mark selection that allows for re-selecting the selected marks as an individual sub-group and specifying the timing and animation attributes for each sub-group. For example, the tick label in the y-axis and all rectangles that share the `Category` attribute can be selected together by *"selector": ".y-axis-label, .rectangle"* (Figure 8 (d, e)), and the re-selected rectangles can be further partitioned by the additional attribute `Country`, if needed. In doing so, CAST+ supports various types of visual marks in the same keyframe group. By setting the duration and delay to each sub-group, users can achieve different pauses between the animation of marks in the same row and between rows.

Semantic Zooming. As our keyframes show the chart's thumbnail, the animation sequence in the specification panel can become too long to be easily accessed. To address this issue, we employ semantic zooming [57] with an aim to preserve the hierarchical grouping structures of keyframe sequences, while maintaining the visibility of visual marks when zoomed out.

To achieve this goal, 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 the 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 at the zoom level 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 hard to read and thus we generate the bounding boxes of them to indicate their existence. In doing so, the bounding boxes provide navigational assistance while directing users to focus on data-encoded visual marks. In our experiment, we found

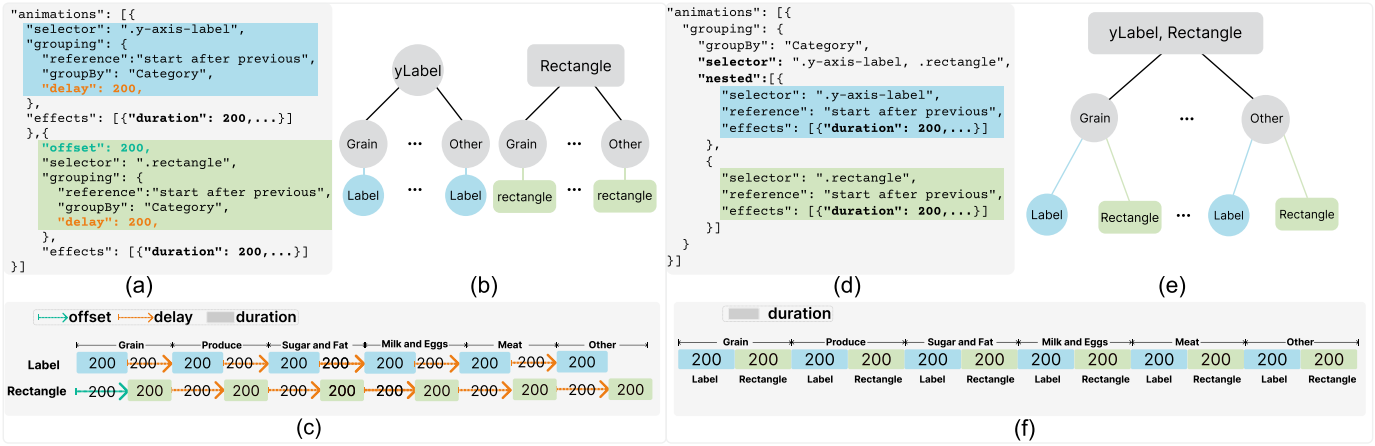


Fig. 8: Comparing the Canis specifications for achieving the animation of the Mekko chart in Scenario 2 without (a) and with the nested selection (d). (b,e) the intermediate mark-unit trees generated during the compilation of the given specifications in (a,d); (c) aligning two timelines corresponding to two types of visual marks requires careful specification of duration and delays; whereas the one in (f) is inherently represented as a linear line.

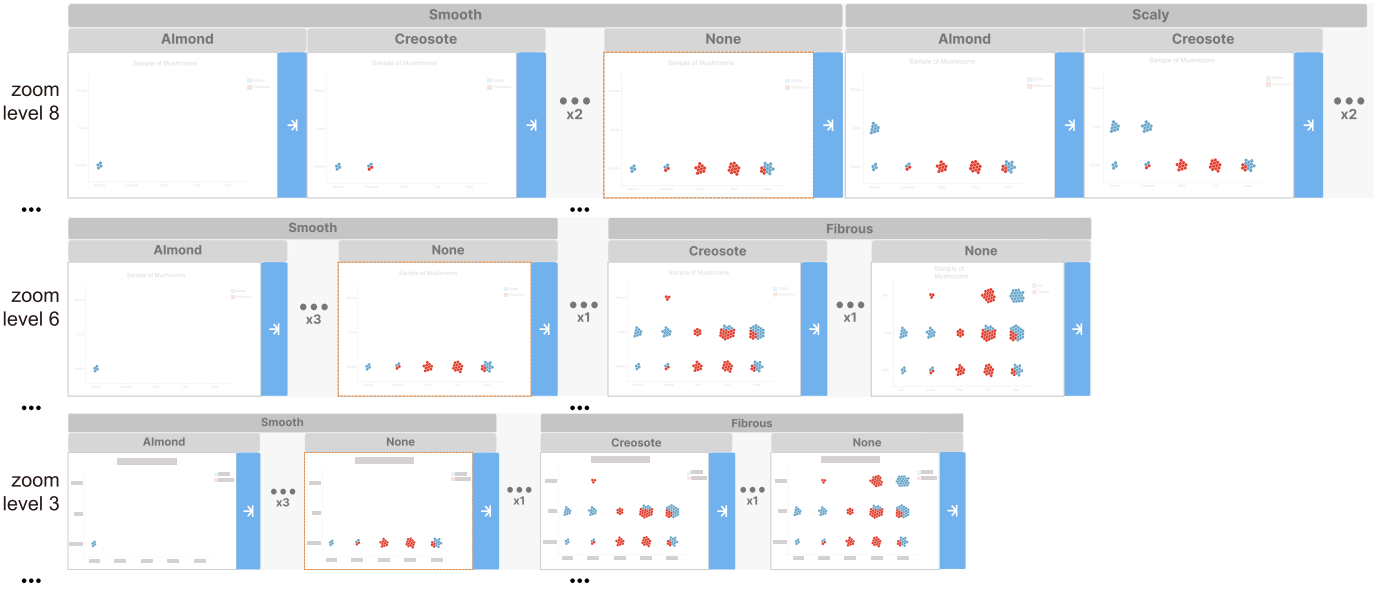


Fig. 9: An example illustrating how the keyframe sequence is changed with semantic zooming on different zoom levels. We present this keyframe sequence on three zoom levels since it shows the same number of keyframes or keyframe groups under level 6, while the size is scaled; The last keyframe in the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

that $l = 8$ is enough in most cases since the depth of the mark unit tree in most animations is less than 5.

F. Implementation Details

CAST+ is an HTML5 web application implemented with TypeScript and Webpack, following the Redux architecture [63]. 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.

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 [64] that re-compiles only a portion of the specification affected by modifications. Given a modified specification resulting from 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.

IV. Evaluation

To demonstrate the expressive power of CAST+, we created a wide variety of chart animations with diverse charts and animation effects. Figure 2 shows eight examples from our gallery and the full gallery can be found on 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 authoring systems (especially designed for communication purposes) [65], [66]. We face similar issues: there are no prior works that allow people to easily create chart animations. In addition, as our system introduces a new visual specification, the authors first need to learn and understand the specification to use the system for authoring chart animations. We therefore decided not to conduct a comparative study, but instead designed a study to evaluate both the visual specification and the system as described below.

After performing a user study with the initial version, i.e., CAST, we addressed the major issues identified from the study in CAST+ by improving auto-completion algorithms and refining the user interaction and interface. To compare CAST and CAST+, we employ the keystroke-level model [67] to analyze the number of user interactions required to reproduce the same animation. This comparison highlights the advantages of CAST+ in reducing low-level interactions and minimizing users' cognitive load

A. User Study

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 [47], [68]–[71].

Participants. We recruited 18 participants (6 females; 12 males) from a local university, all with normal or corrected-to-normal vision. Their ages ranged from 22 to 32, with an average of 25. They major in computer science and most of them have the experience of using video or animation editors, such as Camtasia [72], Adobe After Effect [9], iMovie [73], and Movie Maker [74].

Apparatus. We used a desktop computer with an Intel i7-8700K 3.7GHz CPU and 16GB RAM, with two 27-inch LCD displays placed side-by-side, both running at 1920×1080 resolution. In part 1, we used only 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. 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 (Figure 2 (a-c)) for part 1, covering visual elements and layouts used in the visual specifications, and four animation reproduction tasks (Figure 2 (d-g)) for part 2, encompassing basic interactions in CAST. For both part, the expressiveness and complexity of subsequent tasks increased.

Procedure. After briefly introducing the study goals and procedure, we asked participants to fill out a pre-study questionnaire. We then provided a tutorial on 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 (1: Strongly Disagree and 5: Strongly Agree).

After a 10-minute 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 (1: Strongly Disagree and 5: Strongly Agree). On average, the entire session lasted about 80 minutes (30 for Part 1, 40 for Part 2, and 10 for a break).

Study Results. 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. The responses from the post-study questionnaire (Figure 10 (a)) indicate that the visual representations and the visual specifications are easy to learn (M = 4.7) and easy to understand (M = 4.6).

Seventeen 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

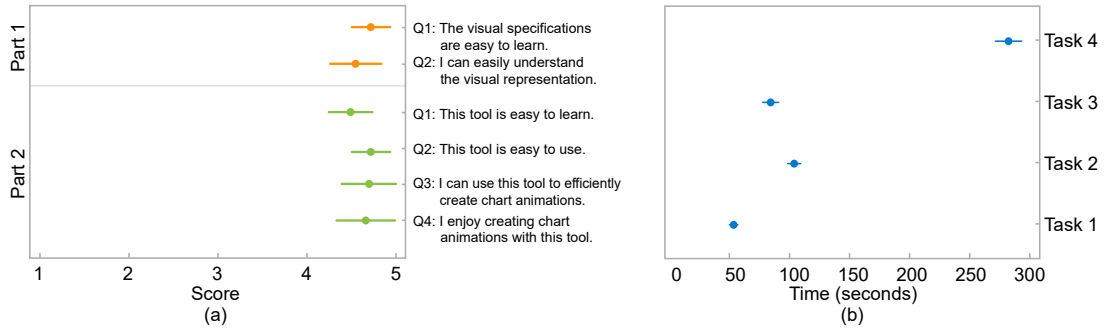


Fig. 10: Results of the user study. (a) The 95% confidence interval in scores to two satisfaction questions in part 1 and four questions in part 2: the higher score the better. (b) The completion time in seconds for four reproduction tasks in part 2: error bars represent the standard deviations.

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 (Figure 10 (b)) was less than two minutes except for Task 4 (282 seconds). 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). Both issues have been resolved in CAST+, where the headers of the nested boxes of keyframes and keyframe groups are used for consistent dragging (see Section III-C) and revised the implementation to avoid unintended interaction feedback.

B. Comparison between CAST and CAST+

To assess the relative interaction complexity of CAST and CAST+, we use both systems to author four animations (Table I), which encompass a variety of visual marks and keyframe sequences. Instead of using click count as a proxy metric [70], we count the number of semantic actions performed by the user to specify an animation [75], as this metric effectively captures cognitive load by focusing on essential authoring actions. There are three stages to specify keyframes: (i) constructing keyframes, (ii) choosing among the suggested next keyframes, and (iii) dragging and dropping keyframes on top of each other to synchronize keyframes. The interactions in three stages often require complex decision-making regarding the information within keyframes, contributing significantly to the cognitive complexity of the task. Note that we did not count repetitive interactions to fine-tune the animation timing and effect properties, assuming that one such interaction yielded the desired result.

We report the number of interactions required at each stage of keyframe manipulation separately in Table I. At stage (i), we report the number of specified keyframes and the interactions, including clicks (to select mark units), drag-and-drops (to move selected marks to the keyframe list), and brushes (to select specific regions of visual marks within the chart panel). At stage (ii), we report the number of specified keyframes and the total number of suggested next keyframes. At stage (iii), we report the number of manipulated keyframes and keyframe groups, along with the number of drag-and-drops. As the process for creating them is the same in both systems, we did not consider the interactions related to the initial keyframes.

Table I shows the comparison results, demonstrating that CAST+ offers lower interaction complexity and supports a broader range of scenarios, whereas CAST cannot support the animation in Scenario 4 (Gantt Chart). Except for Scenario 1 (Faceted Dot Plot), CAST+ requires fewer interactions in Scenarios 2 (Mekko Chart) and 3 (Connected Scatterplot), both of which involve synchronizing keyframe groups of multiple mark types. Since CAST+ allows for constructing keyframes that include various types of marks, it eliminates the need for interactions at stage (iii) and reduces them at Stage (i). In Scenario 3 (Connected Scatterplot), CAST+ completes the entire keyframe sequence based on the keyframes constructed at Stage (i), eliminating suggestions at Stage (ii). Although CAST+ suggests three next-keyframes with different types of marks at stage (ii) in Scenario 2 (Mekko Chart), its total interaction count is still less than that of CAST. For detailed counts of each action, please refer to the supplemental material.

Overall, CAST+ not only reduces the number of low-level interactions but also minimizes the cognitive load on users by automating complex synchronization tasks. This leads to a more efficient workflow, especially when dealing with animations

TABLE I: Comparing the number of keyframe specifications and required actions across three stages—keyframe construction, next-keyframe selection, and keyframe synchronization when authoring chart animations across different charts and systems, where actions include clicking (CL), brushing (BR), and drag-and-drop (DD).

Chart	System	Stage (i)		Stage (ii)		Stage (iii)	
		Specs (Kfs.)	CL+BR+DD	Specs (Kfs)	Kfs Considered	Specs (Kfs.)	DD
Scenario 1 (Faceted dot plot)	CAST	1	2	1	3	0	0
	CAST+	1	2	1	3	0	0
Scenario 2 (Mekko Chart)	CAST	2	5	0	0	1	1
	CAST+	1	2	1	3	0	0
Scenario 3 (Connected Scatterplot)	CAST	3	7	1	2	1	1
	CAST+	2	6	0	0	0	0
Scenario 4 (Gantt Chart)	CAST	Not Supported		Not Supported		Not Supported	
	CAST+	1	2	1	2	1	1

that involve multiple types of marks and intricate timing requirements. Note that the actual number of specification actions may vary based on the user's familiarity and exact order of creation. As such, these numbers should be interpreted as providing a general sense of the interaction complexity.

V. Discussions

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 first sight, since I never thought those charts can be animated in such an 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 with 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 resulting 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."*

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 [65], the approach commonly used for evaluating visualization authoring systems, including Data Illustrator [47], DataInk [69], and Charticulator [70]. We assessed if people could produce chart animations using CAST when provided with a reference animation and a short training (the tutorial and training took about 25 minutes in part 2). 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, and to collect chart animations people create.

We also note that our work can be complemented with further studies. For example, it will be useful 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 more challenging, contribute more 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 [65]. 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 [76]. In 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.

Differences from 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 [70] and D3 [10]. In terms of the types of animation (using DataClip's taxonomy [6]), 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*.

After developing CAST, we enhanced the expressiveness and efficiency in CAST+. In terms of expressiveness, CAST+ allows the creation of animations using various types of visual marks within a keyframe group and offers visual components for binding data attributes to animation duration and delay. For efficiency, we proposed new generalized selection-based auto-completion algorithms for constructing keyframes and keyframe sequences, which leads to fewer interactions for authoring similar animations.

We acknowledge that while CAST+ is a robust tool, it is still unable to generate all the animations that Canis can do due to limitations in the visual specifications. It does not currently support multi-view animations, which are instrumental for facilitating side-by-side comparisons. In addition, it does not allow some essential interactions for authoring *transition* animations, such as dragging the keyframe to specify mark units that animate successively in one chart. This restriction is in place as it could lead to invalid visual mappings during transitions between multiple charts.

VI. 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 author expressive animations through keyframe direct manipulation 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 an 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 of helping people who lack programming skills to easily create chart animations. With five usage scenarios, we illustrated how the visual specification and interface of CAST+ enables a rapid animation creation process. We also explained in detail about the auto-completion based on generalized selection which improves the authoring efficiency and further extends the algorithms to support the animation with various types of visual marks within one keyframe group. 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://canisstudio.github.io/CASTPlus to see the animation, along with our example gallery that demonstrates the system's expressiveness.

Acknowledgments

This work was partially supported by the grants of the National Key R&D Program of China under Grant 2022ZD0160805, the NSFC (No.62132017), the Shandong Provincial Natural Science Foundation (No.ZQ2022JQ32), the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China, and the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT), Artificial Intelligence Graduate School Program, Yonsei University, under Grant RS-2020-II201361.

References

[1] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang, "Canis: A high-level language for data-driven chart animations," *Computer Graphics Forum*, vol. 39, no. 3, pp. 607–617, 2020.

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

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

[4] F. Amini, N. Henry Riche, B. Lee, C. Hurter, and P. Irani, "Understanding data videos: Looking at narrative visualization through the cinematography lens," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2015, pp. 1459–1468.

[5] J. Thompson, Z. Liu, W. Li, and J. Stasko, "Understanding the design space and authoring paradigms for animated data graphics," *Computer Graphics Forum*, vol. 39, no. 3, pp. 207–218, 2020.

[6] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani, "Authoring data-driven videos with dataclips," *IEEE Transactions Visualization and Computer Graphics*, vol. 23, no. 1, pp. 501–510, 2016.

[7] Kiln Enterprises Ltd., "Flourish," https://flourish.studio, 2024, [Accessed 5-August-2024].

[8] Adobe, "Adobe stock," https://stock.adobe.com, 2024, [Accessed 5-August-2024].

[9] Adobe, "Adobe after effects," https://www.adobe.com/products/aftereffects.html, [Online; accessed 5-August-2024].

[10] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Transactions Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[11] H. Wickham, *ggplot2: elegant graphics for data analysis*. Springer, 2016.

[12] T. Ge, B. Lee, and Y. Wang, "CAST: Authoring data-driven chart animations," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.

[13] J. Heer and G. Robertson, "Animated transitions in statistical data graphics," *IEEE Transactions Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1240–1247, 2007.

[14] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko, "Effectiveness of animation in trend visualization," *IEEE Transactions Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1325–1332, 2008.

[15] D. Archambault, H. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *IEEE Transactions Visualization and Computer Graphics*, vol. 17, no. 4, pp. 539–552, 2010.

[16] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe, "A comparative evaluation of animation and small multiples for trend visualization on mobile phones," *IEEE Transactions Visualization and Computer Graphics*, vol. 26, no. 1, pp. 364–374, 2019.

[17] B. Tversky, J. B. Morrison, and M. Betrancourt, "Animation: can it facilitate?" *International journal of human-computer studies*, vol. 57, no. 4, pp. 247–262, 2002.

[18] F. Chevalier, N. H. Riche, C. Plaisant, A. Chalbi, and C. Hurter, "Animations 25 years later: New roles and opportunities," in *Proc. International Conference on Advanced Visual Interfaces*, 2016, pp. 280–287.

[19] C. Ware, "Information visualization: Perception for design," 2004.

[20] C. Ware, E. Neufeld, and L. Bartram, "Visualizing causal relations," in *Proceedings IEEE Symposium on Information Visualization*, vol. 99, no. 1, 1999, pp. 39–42.

[21] L. Bartram and C. Ware, "Filtering and brushing with motion," *Information Visualization*, vol. 1, no. 1, pp. 66–79, 2002.

[22] C. Gonzalez, "Does animation in user interfaces improve decision making?" in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 1996, pp. 27–34.

[23] K. Koffka, "Perception: an introduction to the gestalt-theorie." *Psychological Bulletin*, vol. 19, no. 10, p. 531, 1922.

[24] S. Mateeff, G. Dimitrov, and J. Hohnsbein, "Temporal thresholds and reaction time to changes in velocity of visual motion," *Vision research*, vol. 35, no. 3, pp. 355–363, 1995.

[25] T. Tekušová, J. Kohlhammer, S. J. Skwarek, and G. V. Paramei, "Perception of direction changes in animated data visualization," in *Proceedings of the 5th symposium on Applied perception in graphics and visualization*, 2008, pp. 205–205.

[26] J. A. Greenwood and M. Edwards, "The detection of multiple global directions: Capacity limits with spatially segregated and transparent-motion signals," *Journal of vision*, vol. 9, no. 1, pp. 40–40, 2009.

[27] D. Weiskopf, "On the role of color in the perception of motion in animated visualizations," in *IEEE Visualization 2004*. IEEE, 2004, pp. 305–312.

[28] H. Romat, C. Appert, B. Bach, N. Henry-Riche, and E. Pietriga, "Animated edge textures in node-link diagrams: A design space and initial evaluation," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–13.

[29] A. Chalbi, J. Ritchie, D. Park, J. Choi, N. Roussel, N. Elmqvist, and F. Chevalier, "Common fate for animated transitions in visualization," *IEEE Transactions Visualization and Computer Graphics*, vol. 26, no. 1, pp. 386–396, 2019.

[30] D. Fisher, "Animation for visualization: opportunities and drawbacks," 2010, vol. 19, ch. 19, pp. 329–352.

[31] M. Shanmugasundaram, P. Irani, and C. Gutwin, "Can smooth view transitions facilitate perceptual constancy in node-link diagrams?" in *Proceedings of Graphics Interface*, 2007, pp. 71–78.

[32] P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete, "Temporal distortion for animated transitions," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2009–2018.

[33] F. Chevalier, P. Dragicevic, and S. Franconeri, "The not-so-staggering effect of staggered animated transitions on visual tracking," *IEEE Transactions Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2241–2250, 2014.

[34] F. Du, N. Cao, J. Zhao, and Y.-R. Lin, "Trajectory bundling for animated transitions," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2015, pp. 289–298.

[35] Y. Wang, D. Archambault, C. E. Scheidegger, and H. Qu, "A vector field design approach to animated transitions," *IEEE Transactions Visualization and Computer Graphics*, vol. 24, no. 9, pp. 2487–2500, 2017.

[36] L. Grammel, C. Bennett, M. Tory, and M.-A. D. Storey, "A survey of visualization construction user interfaces." in *EuroVis (Short Papers)*, 2013.

[37] N. Kramer, "10 best ui animation libraries for beginners," https://daily.dev/blog/10-best-ui-animation-libraries-for-beginners-2024, 2024, [Online; accessed 8-August-2024].

[38] R. Rischpater and D. Zucker, "Introducing qt quick," in *Beginning Nokia Apps Development*. Springer, 2010, pp. 139–158.

[39] D. Ren, B. Lee, and T. Höllerer, "Stardust: Accessible and transparent gpu support for information visualization rendering," *Computer Graphics Forum*, vol. 36, no. 3, pp. 179–188, 2017.

[40] T. L. Pedersen and D. Robinson, "A grammar of animated graphics — gganimate," https://gganimate.com, [Accessed 5-August-2024].

[41] H. Wickham, "A layered grammar of graphics," *Journal of Computational and Graphical Statistics*, vol. 19, no. 1, pp. 3–28, 2010.

[42] Y. Kim and J. Heer, "Gemini: A grammar and recommender system for animated transitions in statistical graphics," *IEEE Transactions Visualization and Computer Graphics*, vol. 27, no. 2, pp. 485–494, 2020.

[43] Y. Kim and J. Heer, "Gemini 2: Generating keyframe-oriented animated transitions between statistical graphics," in *Proc. IEEE Conf. on Visualization*. IEEE, 2021, pp. 201–205.

[44] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan, "Animated vega-lite: Unifying animation with a grammar of interactive graphics," *IEEE Transactions Visualization and Computer Graphics*, vol. 29, no. 1, pp. 149–159, 2022.

[45] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Transactions Visualization and Computer Graphics*, vol. 23, no. 1, pp. 341–350, 2016.

[46] J. R. Thompson, Z. Liu, and J. Stasko, "Data animator: Authoring expressive animated data graphics," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–18.

[47] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko, "Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2018, p. 123.

[48] J. Heer, M. Agrawala, and W. Willett, "Generalized selection via interactive query relaxation," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 959–968.

[49] K. Marriott and B. Meyer, *Visual language theory*. Springer Science & Business Media, 1998.

[50] J. Rumbaugh, I. Jacobson, and G. Booch, "The unified modeling language," *Reference manual*, 1999.

[51] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A system for query, analysis, and visualization of multidimensional relational databases," *IEEE Transactions Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52–65, 2002.

[52] J. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Transactions Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1137–1144, 2007.

[53] G. G. Méndez, M. A. Nacenta, and S. Vandenheste, "iVoLVER: Interactive visual language for visualization extraction and reconstruction," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2016, pp. 4073–4085.

[54] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Computer Graphics Forum*, vol. 33, no. 3, pp. 351–360, 2014.

[55] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 3363–3372.

[56] J. D. Eisenberg and A. Bellamy-Royds, *SVG Essentials: Producing Scalable Vector Graphics with XML*. " O'Reilly Media, Inc.", 2014.

[57] B. B. Bederson and J. D. Hollan, "Pad++ a zooming graphical interface for exploring alternate interface physics," in *Proc. ACM Symposium on User Interface Software and Technology*, 1994, pp. 17–26.

[58] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92–99, 1992.

[59] H. Fairfield, "Driving shifts into reverse," 2009. [Online]. Available: https://archive.nytimes.com/www.nytimes.com/imagepages/2010/05/02/business/02metrics.html

[60] B. Bach, M. Stefaner, J. Boy, S. Drucker, L. Bartram, J. Wood, P. Ciuccarelli, Y. Engelhardt, U. Koeppen, and B. Tversky, "Narrative design patterns for data-driven storytelling," in *Data-Driven Storytelling*. AK Peters/CRC Press, 2018, pp. 125–152.

[61] S. E. Palmer, *Vision science: Photons to phenomenology*. MIT press, 1999.

[62] Y. Kim, K. Wongsuphasawat, J. Hullman, and J. Heer, "Graphscape: A model for automated reasoning about visualization similarity and sequencing," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2017.

[63] H. Singh and M. Bhatt, *Learning Web Development with React and Bootstrap*. Packt Publishing Ltd, 2016.

[64] S. P. Reiss, "An approach to incremental compilation," *ACM SIGPlan Notices*, vol. 19, no. 6, pp. 144–156, 1984.

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

[66] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. Stasko, J. Thompson, M. Brehmer, and Z. Liu, "Critical reflections on visualization authoring systems," *IEEE Transactions Visualization and Computer Graphics*, vol. 26, no. 1, pp. 461–471, 2019.

[67] S. K. Card, T. P. Moran, and A. Newell, "The keystroke-level model for user performance time with interactive systems," *Communications of the ACM*, vol. 23, no. 7, pp. 396–410, 1980.

[68] D. Ren, M. Brehmer, B. Lee, T. Höllerer, and E. K. Choe, "ChartAccent: Annotation for data-driven storytelling," in *Proc. IEEE Pacific Visualization Symposium*, 2017, pp. 230–239.

[69] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor, "Dataink: Direct and creative data-oriented drawing," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2018.

[70] D. Ren, B. Lee, and M. Brehmer, "Charticulator: Interactive construction of bespoke chart layouts," *IEEE Transactions Visualization and Computer Graphics*, vol. 25, no. 1, pp. 789–799, 2018.

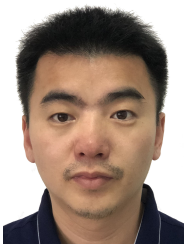
[71] J. E. Zhang, N. Sultanum, A. Bezerianos, and F. Chevalier, "Dataquilt: Extracting visual elements from images to craft pictorial visualizations," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[72] TechSmith, "Camtasia: Screen recorder & video editor," https://www.techsmith.com/video-editor.html, 2024, [Accessed 5-August-2024].

[73] Apple, "imovie," https://www.apple.com/imovie/, 2024, [Accessed 5-August-2024].
[74] Microsoft, "Windows Movie Maker," http://moviemaker.support/, 2024, [Accessed 5-August-2024].
[75] S. Wilson and R. Mihalcea, "Measuring semantic relations between human activities," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017, pp. 664–673.
[76] Y. Yang, H. S. Solgaard, and J. Ren, "Does positive framing matter? an investigation of how framing affects consumers' willingness to buy green electricity in denmark," *Energy research & social science*, vol. 46, pp. 40–47, 2018.

Yuancheng Shen was working toward his M.S. degree from Shandong University, where he works with Prof. Yunhai Wang in the IDEAS Lab. His research interests include data visualization, data-driven animation, and interactive data exploration and analysis system.

Yue Zhao is working toward a fourth-year PhD. degree with the School of Computer Science and Technology, Shandong University. His research interests include information visualization and human-computer interaction.



Yunhai Wang (Member, IEEE) is a professor in the School of Information at Renmin University of China. He serves as the associate editor of the IEEE Transactions on Visualization and Computer Graphics, Computer Graphics Forum, and IEEE Computer Graphics and Applications. His interests include scientific visualization, information visualization, and computer graphics.

Tong Ge is a research scientist at Beke. He received his Ph.D. degree from Shandong University, where he works with Prof. Yunhai Wang and Prof. Baoquan Chen in the IDEAS Lab. His research interests are data visualization, application system, and machine learning. Specifically, he has been working on programming languages (e.g. Canis) and authoring tools (e.g. CAST) for data-driven chart animation.

Haoyan Shi is currently working toward a second-year master's degree with the School of Computer Science and Technology, Shandong University. His research interests include information visualization and data-driven animation.

Bongshin Lee is a professor at Yonsei University. She conducts research on human-data interaction and human-computer interaction, with an overarching goal to empower everyone to achieve their goals by leveraging data, visualization, and technological advancements. Lee explores innovative ways to help people interact with data, by supporting data collection, reflection, and analysis, as well as data-driven communication. She received her PhD from the University of Maryland at College Park.