Animated Vega-Lite: Unifying Animation with a Grammar of Interactive Graphics

Jonathan Zong*, Josh Pollock*, Dylan Wootton, Arvind Satyanarayan

Abstract— We present Animated Vega-Lite, a set of extensions to Vega-Lite that model animated visualizations as time-varying data queries. In contrast to alternate approaches for specifying animated visualizations, which prize a highly expressive design space, Animated Vega-Lite prioritizes unifying animation with the language's existing abstractions for static and interactive visualizations to enable authors to smoothly move between or combine these modalities. Thus, to compose animation with static visualizations, we represent time as an *encoding channel*. Time encodings map a data field to animation keyframes, providing a lightweight specification for animations without interaction. To compose animation and interaction, we also represent time as an *event stream*; Vega-Lite selections, which provide dynamic data queries, are now driven not only by input events but by timer ticks as well. We evaluate the expressiveness of our approach through a gallery of diverse examples that demonstrate coverage over taxonomies of both interaction and animation. We also critically reflect on the conceptual affordances and limitations of our contribution by interviewing five expert developers of existing animation grammars. These reflections highlight the key motivating role of in-the-wild examples, and identify three central tradeoffs: the language design process, the types of animated transitions supported, and how the systems model keyframes.

Index Terms—Information visualization, Animation, Interaction, Toolkits, Systems, Declarative Specification

1 Introduction

Rapid prototyping is critical to the visualization authoring process. When making an explanatory graphic, rapid prototyping allows a visualization author to evaluate candidate designs before committing to refining one in detail. For exploratory data analysis, rapid prototyping is equally key as visualization is just one part of a broader workflow, with analysts focused on producing and analyzing a chart to yield insight or seed further analysis. However, consider the friction of visualizing faceted data: an author might choose between depicting facets as a small multiples display, on-demand via interaction (e.g., dynamic query widgets), or played sequentially via animation. These designs make different trade-offs between time and space and, as a result, research results suggest they afford readers different levels of clarity, time commitment, and visual interest [33]. Despite these differences, the designs express a shared goal — to visualize different groupings of the data — and a visualization author might reasonably expect to be able to easily move between the three to make the most appropriate choice.

Unfortunately, existing visualization toolkits can present a highly viscous [44] specification process when navigating this time-space trade-off. One class of toolkits supports either interaction or animation, but not both. Such systems include Vega [38] and Vega-Lite [36] — which offer interaction primitives in the form of *signals* and *selections* but do not provide abstractions for animation — as well as gganimate [43], Data Animator [46], Canis / CAST [9, 10], and Gemini/Gemini^2 [19, 20] — which express animation in terms of transitions between discrete visualization states known as *keyframes* but do not provide treatment for interaction. As a result, these systems force visualization authors to prematurely commit [44] to either an interaction- or animation-friendly abstraction when choosing their prototyping tool, and thus limit authors' ability to explore alternative designs. A second class of toolkits (including D3 [3] and Plotly [1]) support both modalities but do so via largely distinct abstractions (namely, *transitions* or *frames* for animation, and *event handlers* or a typology of techniques for interaction). Thus, an author must often either restructure or rewrite their specifications to consider interaction and animation in parallel.

- *Jonathan Zong and Josh Pollock are co-first authors.*
- *The authors are with MIT CSAIL. E-mails: {jzong, jopo, dwootton, arvindsatya}@mit.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

In this paper, we present Animated Vega-Lite: extensions to Vega-Lite to support data-driven animation. Its design is motivated by the key insight that interaction and animation are parallel concepts (Sect. 3). Whereas interactions transform data (e.g. filtering) and update visual properties (e.g. re-coloring marks) in response to *user* input, animations do the same in response to a *timer*. From this perspective, interactive and animated visualization techniques occupy a spectrum of dynamic, event-driven behaviors. Thus, with Animated Vega-Lite, animated visualizations (like their interactive counterparts) are modeled as time-varying data queries — an approach that allows us to provide a *unified* set of abstractions for static, interactive, and animated visualizations.

Animated Vega-Lite offers two abstractions of time that allow animations to compose with Vega-Lite's existing grammars of static and interactive visualizations (Sect. 4). From the perspective of interaction, time is an *event stream*: a source of events analogous to `clicks` and `keypresses` produced by a user. These events drive Vega-Lite *selections*, which apply dynamic data queries to visual encodings. Thus, by modeling time as an event stream, users can seamlessly specify and move between interactive and animated behavior in the same specification. From the perspective of Vega-Lite's grammar of graphics, time is an *encoding channel*. Just as `x` and `y` encodings map data values to spatial positions measured in pixels, a `time` encoding maps data values to temporal positions measured in elapsed milliseconds. Compared to the event stream abstraction, the encoding channel abstraction is lighter-weight, but less expressive. This allows a visualization author to get started quickly with an animated chart and to move easily between an animated and a faceted visualization by switching a `time` channel for a `row` or `column` one. And, for added customizability, users can always turn a time-as-encoding specification into a time-as-event-stream one.

We implement a prototype compiler that synthesizes a low-level Vega specification with shared reactive logic for interaction and animation (Sect. 5). Following best practices [32], we assess our contribution with multiple evaluation methods. Through a diverse example gallery (Sect. 6), we demonstrate that Animated Vega-Lite covers much of Yi et al.'s interaction taxonomy [51] and Heer & Robertson's animation taxonomy [12] while preserving Vega-Lite's low viscosity and systematic generativity. We also interview five expert developers of four existing animated visualization grammars [9, 10, 19, 20, 42, 46] to critically reflect [35] on the tradeoffs, conceptual affordances, and limitations of our system (Sect. 7). We discuss the important role example visualizations play in grammar design and analyze three areas of tradeoffs: the language design process, support for animations within vs. between encodings, and models of animation keyframes.

2 Related Work

Our contribution is motivated by perceptual work on the value of combining interaction and animation, and is informed by the design of existing toolkits for authoring animated data visualizations.

2.1 Animation in Information Visualization

In a classic 2002 paper, Tversky et al. [47] question the efficacy of animated graphics. In reviewing nearly 100 studies comparing static and animated graphics, the authors were unable to find convincing cases where animated charts were strictly superior to static ones. Visualization researchers have since contributed a body of studies that have identified reasons to be both optimistic and cautious about the value of animation in visualization. For instance, several studies have demonstrated advantages when animating chart transitions [5, 7, 12, 18] or directly animating data values to convey uncertainty [13, 17]. However, these studies have also echoed concerns from Tversky et al. that animations are often too complex or fast to be perceived accurately — for instance, Robertson et al. found that animated trend visualizations are outperformed by static small multiples displays [33].

To ameliorate these limitations of animation, Tversky et al. suggest composing animation with interactivity, particularly through techniques that allow reinspection or focusing on subsets of depicted data. Robertson et al. began to probe this question by testing an interactive alternative alongside the static and animated stimuli — here, clicking an individual mark adds an overlaid line that depicts its trajectory over time. They find that although participants are no more accurate under this interactive condition, they perform faster when using this visualization for data analysis [33]. In follow-up work, Abukhodair et al. [2] further contextualize Robertson's results, finding that interactive animation can be effective and significantly more accurate than animation alone when users want to drill down into the data or have specific questions about points of interest. More recent results are similarly promising: in eye-tracking studies, Greussing et al. [11] find that interactive animated graphics not only received more attention than static or interactive-only equivalents, but these charts also produced higher knowledge acquisition in participants. The authors believed that the enhanced affects on memory and performance resulted from an increase in engagement and attention on the visualization, which is in line with additional research on attention [4]. Our work is motivated by these results. By providing a *unified* abstraction of interaction and animation, Animated Vega-Lite allows analysts to rapidly switch between the two modalities, or compose them together to best suit their needs. Moreover, as our abstractions preserve Vega-Lite's generative properties, we believe our contribution lowers the threshold for conducting future such studies by allowing researchers to more systematically isolate, vary, and compare individual interaction and animation techniques.

2.2 Authoring Interaction and Animation

In Sect. 3.1, we describe the conceptual similarities between Animated Vega-Lite and Functional Reactive Programming (FRP). Moreover, in Sect. 7 we conduct a detailed comparison between Animated Vega-Lite and gganimate [42], Data Animator [46], Gemini/Gemini^2 [19, 20], and Canis/CAST [9, 10]. Here, we instead survey other systems for authoring interaction and animation that have informed our approach.

Visualization toolkits such as D3 [3], Plotly [1], and Matplotlib [14] offer a number of facilities for authoring and composing interaction and animation including typologies of techniques (e.g., brushing, hovering, and animation frames) through to event callbacks and transition functions. Technique typologies can help foster a rapid authoring process, allowing designers to easily instantiate common techniques, but also present a sharp *abstraction* cliff [44]. If designers wish to produce more custom interaction or animation techniques, they must turn to an entirely different notation: authoring low-level, imperative event callbacks or transition functions. This abstraction cliff also increases the *viscosity* of the authoring process [44]. For instance, to switch between the static, interactive, and animated displays of faceted data described in the introduction using D3 would involve restructuring the specification code in non-trivial ways — a problem that is exacerbated if HTML templates are used to generate the SVG rather than the `d3-selection`, as is increasingly the case when working with modern frontend frameworks such as Svelte, Vue, or React.

In contrast, Animated Vega-Lite, like its predecessor, prioritizes concise high-level declarative specification. As Sect. 3 describes, users can make atomic edits (i.e., changing individual keywords, or adding a localized handful of lines of specification code) to rapidly explore designs across the three modalities. The tradeoff, however, is one of expressiveness. Animated Vega-Lite users are limited to composing language primitives; while these primitives are sufficient to broadly cover interaction and animation taxonomies (Sect. 6), their expressive range will necessarily be smaller than their lower-level counterparts.

Example technique	Interaction intent [51]	Animation type [12]
Conditional encoding	Select	—
Panning	Explore	View transformation
Zooming	Abstract / Elaborate	View transformation
Axis re-scaling	Reconfigure	Substrate transformation
Axis sorting	Reconfigure	Ordering
Filtering	Filter	Filtering
Enter/exit	Explore	Timestep
Multi-view	Connect	—
Changing encodings	Encode	Visualization change, Data schema change

Table 1. Techniques common to interaction and animation taxonomies.

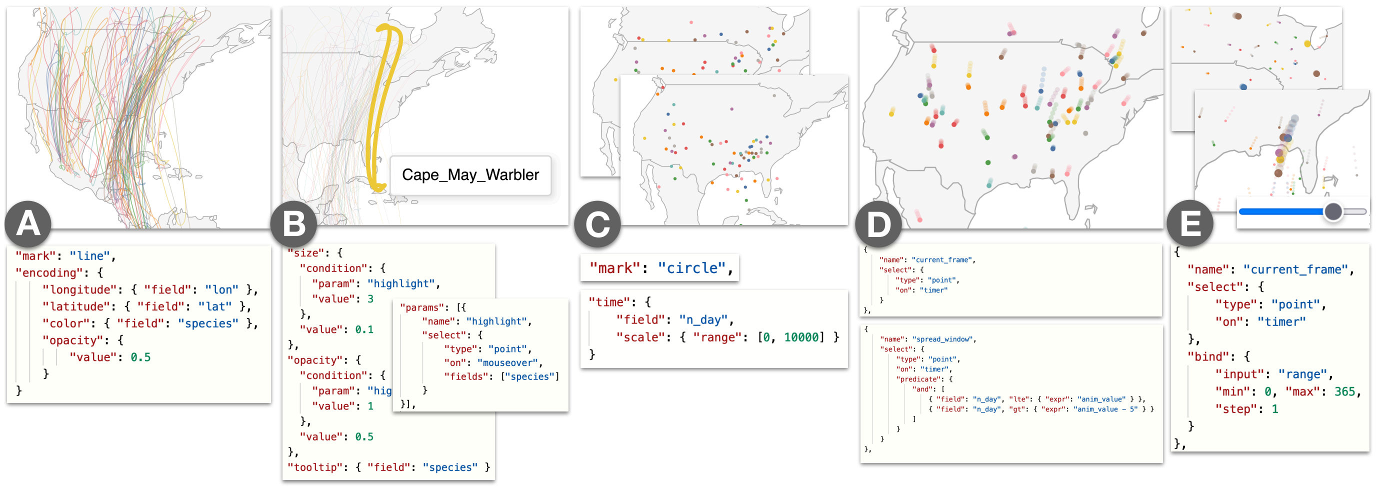
3 Motivation: Unifying Interaction and Animation

In this section, we discuss similarities between interaction and animation that we observe. These similarities drive our design decisions, allowing us to extend Vega-Lite with only minimal additional language primitives, and yielding a low-viscosity grammar that makes it easy to switch between static, interactive and animated modalities.

3.1 Conceptually Bridging Interaction and Animation

We observe that interaction and animation share conceptual similarities at both low and high levels of abstraction. At a low level of abstraction, Functional Reactive Programming (FRP) languages like Flapjax [26] and Fran [8], as well as FRP-based visualization toolkits like Vega [37], have shown that interaction and animation can both be modeled as *event streams*. The Vega example gallery demonstrates how this unified abstraction offers *consistency*, with similar semantics expressed through similar syntactic forms [44]. Namely, the gallery recreates the Gapminder global health scatter plot, originally an animated visualization produced by Hans Rosling [34], but as an interactive visualization driven by the DimpVis direct manipulation technique [21]. We observe that, although it would be tedious to do manually, a user could convert this interactive visualization back to the original animated one by replacing signals near the top of the dataflow, which react to incoming drag events, with signals that respond to timer events instead: where these signals map the drag event's position to a year value, the timer signals would simply emit the next year value on each event. The rest of the downstream reactive logic would remain unchanged. However, as the Vega authors found [38], additional language design is necessary to ensure FRP primitives compose together with grammar of graphics constructs and to facilitate higher-level authoring of dynamic visualizations.

To analyze conceptual similarities between interaction and animation at a higher-level of abstraction, we look to Yi et al. [51] and Heer and Robertson [12] that taxonomize techniques for each modality respectively. These taxonomies are defined by drawing on example visualizations, and although they have been defined separately, share many motivating techniques (Table 1). For example, Heer and Robertson cite



```
"mark": "line",
"encoding": {
    "longitude": { "field": "lon" },
    "latitude": { "field": "lat" },
    "color": { "field": "species" },
    "opacity": {
        "value": 0.5
    }
}
```

```
"size": {
  "condition": {
    "param": "highlight",
    "value": 3
  },
  "value": 0.1
},
"opacity": {
  "condition": {
    "param": "hig
    "value": 1
  },
  "value": 0.5
},
"tooltip": { "field": "species" }
```

```
"params": [{
    "name": "highlight",
    "select": {
        "type": "point",
        "on": "mouseover",
        "fields": ["species"]
    }
}],
```

```
"mark": "circle",
```

```
"time": {
    "field": "n_day",
    "scale": { "range": [0, 10000] }
}
```

```
{
    "name": "current_frame",
    "select": {
        "type": "point",
        "on": "timer"
    }
},
```

```
{
    "name": "spread_window",
    "select": {
        "type": "point",
        "on": "timer",
        "predicate": {
            "and": [
                { "field": "n_day", "lte": { "expr": "anim_value" } },
                { "field": "n_day", "gt": { "expr": "anim_value - 5" } }
            ]
        }
    }
},
```

```
{
    "name": "current_frame",
    "select": {
        "type": "point",
        "on": "timer"
    },
    "bind": {
        "input": "range",
        "min": 0, "max": 365,
        "step": 1
    }
},
```

Fig. 1. An analyst's workflow with Animated Vega-lite. A) Static visualization of bird migrations. B) Adding interaction to hover over a migration path and view a tooltip. C) Switching from static lines to animated circle marks. D) Adding animated path trails for the previous 5 days. E) Adding an interactive slider to scrub through the animation.

panning as an example of *view transformation* because it changes the reader's viewpoint while leaving data schemas and encodings intact. Yi et al. also consider panning, categorizing it as an example of an *explore* interaction, because it involves showing a new subset of data as points shift in and out of the viewport. Zooming, another example of *view transformation*, is also described as an *abstract/elaborate* interaction because it can be used to show data at different levels of detail. As we show in Table 1, we observe substantial overlap in techniques referenced by both taxonomies. Though *select* interactions lack an explicitly defined corresponding animation type, conditional encoding is a commonly used technique in animated visualizations. Similarly, though there is no corresponding category in Heer and Robertson's taxonomy for *connect* interactions, animations applied to shared backing data across multiple views can fulfill the same purpose of highlighting relationships between related points.

3.2 Low-Viscous Authoring: An Example Usage Scenario

A unified abstraction for static, interaction and animation also promotes a low-viscous authoring process (i.e., being able to easily switch between modalities, or compose them together). To illustrate the affordances of this approach, we present an example walkthrough following Imani, an orthonologist, as she plans a new birdwatching expedition. Imani has a bird migration dataset comprising the average latitudes and longitudes for a variety of bird species, for every day of the year [22]. To ensure a productive trip, Imani wants uncover how migration patterns correspond to different times of the year and geographic regions.

Static (Fig. 1A). Imani begins her analysis with a static visualization to get an overview of the dataset. She plots a map, and visualizes migration paths using line marks: each bird species is depicted as a single, uniquely-colored line, connecting the individual daily points along their given latitudes and longitudes. However, Imani is quickly overwhelmed as the size of the dataset produces too many overlapping lines for this static view to be useful, even after adjusting mark opacity.

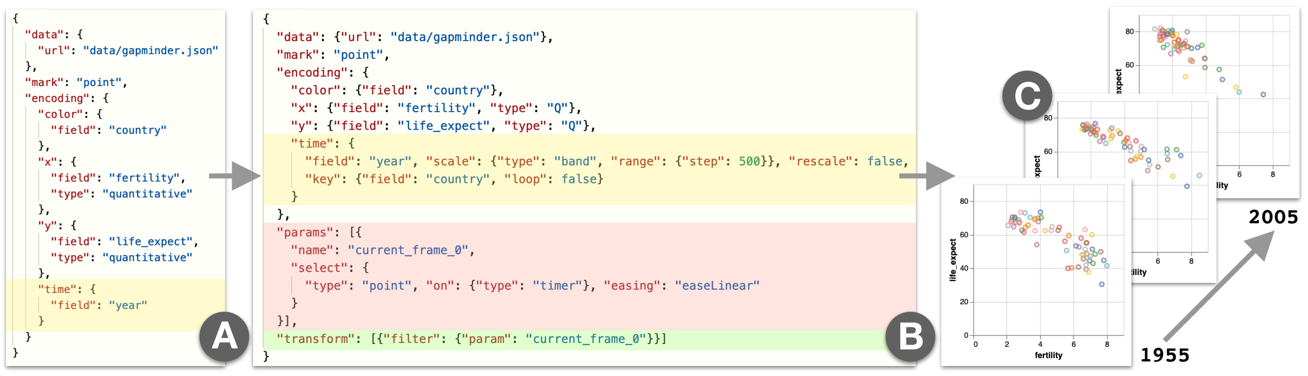
Interactive (Fig. 1B). To pick out individual bird species, and begin a cycle of generating and answering hypotheses, Imani thinks to layer some interactivity on the static display. She adds a *point selection* named `highlight` and driven by *mouseover* events. By default this selection is populated with the data tuple underneath the mouse cursor, and additional tuples are added or toggled when the `shift` modifier key is pressed. Imani writes a *conditional encoding* to interactively adjust mark appearance: selected paths are drawn at full opacity and in a larger size, while unselected paths are drawn with lower opacity and at a smaller size. Thus, as Imani moves her mouse across the visualization, she is able to better trace individual paths, and she adds a *tooltip* encoding channel to surface and note species' names.

This interactive view gives Imani a better sense of migration paths. But, to be able to plan her expedition, she needs to understand where different bird species may be on any given day. Until this point, Imani has used vanilla Vega-Lite abstractions. In the subsequent steps, we show how features of Animated Vega-Lite help Imani deepen her analysis.

Time Encoding Channel (Fig. 1C). Imani swaps to a circle mark and maps `day` (a field that encodes the day of the year from 0 to 365) to the new *time encoding channel*. With these two edits, each bird species is drawn as a circle indicating its location on a particular day, and the visualization animates through `day` values. Imani can now follow the path bird species travel over the course of a year.

Time Event Stream (Fig. 1D). Imani, however, is keenly aware that her dataset only contains average values for each species. Birds tend to appear at a given location within a small window of time around the average day in the dataset. Thus, to ensure she does not make an erroneous conclusion, Imani wants to visualize this variability as a path trail. To do so, she adds a new point selection named `spread_window`, which contains a custom *predicate* — a function that identifies which data tuples should be considered as falling within the selection. In this case, Imani writes a predicate to select data from the five days previous to the current day. She does this by writing inequality expressions referencing the reserved name `anim_value`, which stores the current data value of the animation. In contrast to the existing `highlight` point selection, which is updated on user input events, `spread_window` is instead populated and re-populated on every *timer* tick. She uses `spread_window` to dynamically filter the circle marks, ensuring only data values that lie within the selection are displayed and animated. To visually distinguish the current day's points, she also elaborates the time encoding into an explicit selection called `current_frame` and uses it to drive a conditional opacity encoding. She renders current points at full opacity while rendering the trailing points at less opacity.

Composing Interaction + Animation (Fig. 1E). While watching this path-trail animation, Imani notices that a cluster of birds appear to visit Pensacola, Florida during late March and notes this region as a potential location for her expedition. However, before she lets her colleagues know, she wants to investigate the migration patterns of the birds that come through the area — if these species tend to co-locate in other parts of the world, there is less of a reason for birders to travel to Pensacola specifically. To answer this question, Imani needs finer control over the animation state. She binds the `current_frame` selection to an interactive range slider, and can now toggle between animating and interactively sliding the `day` field. She scrubs the slider to the day when the birds pass through Pensacola, and to track these species in the visualization, she modifies the interactive `highlight` selection to fire on click instead of hover. Imani multi-selects (i.e., clicking with the `shift` key pressed) the birds that pass through the area, and then scrubs to a different day. Here Imani can see that these



```
{
  "data": {
    "url": "data/gapminder.json"
  },
  "mark": "point",
  "encoding": {
    "color": {
      "field": "country"
    },
    "x": {
      "field": "fertility",
      "type": "quantitative"
    },
    "y": {
      "field": "life_expect",
      "type": "quantitative"
    },
    "time": {
      "field": "year"
    }
  }
}
```

```
{
  "data": {"url": "data/gapminder.json"},
  "mark": "point",
  "encoding": {
    "color": {"field": "country"},
    "x": {"field": "fertility", "type": "Q"},
    "y": {"field": "life_expect", "type": "Q"},
    "time": {
      "field": "year", "scale": {"type": "band", "range": {"step": 500}}, "rescale": false,
      "key": {"field": "country", "loop": false}
    }
  },
  "params": [{
    "name": "current_frame_0",
    "select": {
      "type": "point", "on": {"type": "timer"}, "easing": "easeLinear"
    }
  }],
  "transform": [{"filter": {"param": "current_frame_0"}}]
}
```

Fig. 2. Animated Vega-Lite specification of the influential Gapminder animation [34]. (A) A minimal specification using only time encoding. (B) The same specification elaborated to show default encoding properties and a default selection. (C) Selected keyframes from the resulting animation.

birds come from 5 unique nesting sites across the mid-west US to eastern Canada. This is promising as it indicates that these species uniquely overlap in Pensacola, making it a prime viewing destination.

Summary. With Animated Vega-Lite, Imani was able to move between static, interactive, and animated visualizations through a series of atomic edits or otherwise localized changes rather than larger-scale refactoring or restructuring of code. Moreover, we have extended Vega-Lite's high-level affordances to animation: Imani was able to express animation as data selections and transformations, rather than manipulating keyframes or specifying transition states; and, the Animated Vega-Lite compiler synthesized appropriate defaults and underlying machinery for the animation to unfold correctly. Finally, as Animated Vega-Lite offers a unified abstraction, Imani was able to reuse Vega-Lite's existing primitives to author mixed interactive-animated visualizations as well as custom techniques without the need for special-purpose functions — e.g., combining animations with on-click highlighting and composing selections with a window data transform to draw trailing marks, rather than using a `shadow` function as with gganimate.

4 A Grammar of Animation in Vega-Lite

In Animated Vega-Lite, users specify animation using a *time encoding channel* and *timer-driven selections*. Time encodings provide a light-weight way to convert faceted static visualizations into animations. To further customize the animation design or easily add interaction, users can specify animations as selections instead. Selections express dynamic data queries, and are now populated either by input events (as with vanilla Vega-Lite) or, now, via timer ticks. Defined selections can then be used to drive data transformations, scale functions, or conditionally encode visual properties. Our animation model expressively extends existing abstractions for static and interactive visualizations while minimally increasing language surface area and complexity.

4.1 Time Encoding Channel

In Vega-Lite, encodings determine how data values map to the visual properties of a mark (also known as channels). Vega-Lite includes two channels for spatial position, `x` and `y`. Animated Vega-Lite adds a new channel for temporal position, called `time`. A user specifies a time encoding by providing a `field` property, which is a string of the name of a data column. The field can be any measure type with a sort order (quantitative, temporal, ordinal), and does not necessarily need to represent a timestamp. The system uses distinct values from this column to group data rows into temporal facets called *keyframes*. Over the duration of the animation, each keyframe is shown sequentially.

Fig. 2A shows the Animated Vega-Lite specification for Rosling's Gapminder animation [34]. The time encoding, highlighted in yellow, maps the dataset's `year` field to the time encoding channel. The system uses the distinct values of `year` to group rows into keyframes. In other words, there is one keyframe per possible value of `year` in the dataset (i.e. `1955, 1960, 1965, ..., 2005`) (Fig. 2C).

4.1.1 Key Field

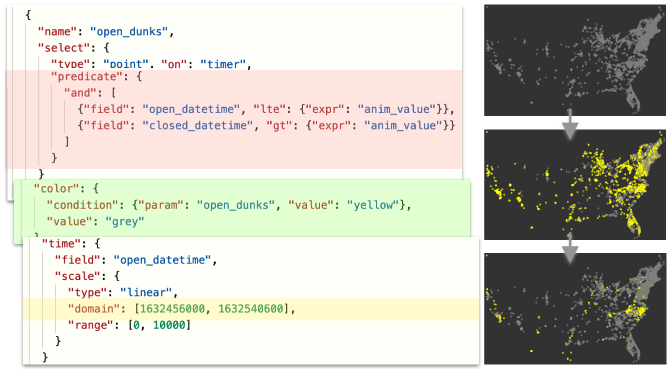
In-betweening, more commonly called *tweening*, is a standard animation technique that involves generating additional frames to smoothly transition between two keyframes. By adding tweening, the animation will give the visual impression of continuous change over time even when data represents discrete measurements. In data visualization, tweening takes on additional meaning as it requires generating and interpolating between values that are not present in the dataset. In Animated Vega-Lite, to specify tweening between keyframes, the user specifies a `key` property in the time encoding channel, which references a field name. This key field is used to group rows together across keyframes. For two given successive keyframes, rows that share the same value for the key field are treated as the start and end states for a single mark instance. Key values should be unique within a keyframe to prevent ambiguity; otherwise, a single mark instance might have multiple start or end states, resulting in undefined behavior. If the user does not specify a key field, the Animated Vega-Lite compiler attempts to infer a sensible default based on the mark type and other specified categorical channels such as `color` or `detail` — an approach that follows Vega-Lite's existing inferences.

In the Gapminder example, Fig. 2B shows the Gapminder spec from Fig. 2A with default values specified explicitly. Here, `country` is used as the default key field as it is also encoded on the `color` encoding channel. Consider the successive keyframes with `year` values 1955 and 1960. For each year, each scatterplot point is identified by a unique `country` value. Therefore, to tween from 1955 to 1960, the system interpolates the two rows for each country to produce the corresponding in-between point at each animation frame.

4.1.2 Time Scale

An encoding uses a scale function to map from the data domain to a visual range. For spatial encoding channels, this range is measured in pixels relative to the bounding box of the rendered visualization. For the time encoding channel, we measure the range in milliseconds elapsed from the start of the animation. Users specify the timing of the animation using a time scale (for example, by specifying either an overall animation duration or the amount of time between keyframes as a `step`). As with existing encoding channels, if a scale is not specified by the user, Vega-Lite infers default scale properties. By default, scales for the time encoding channel use the unique values of the backing field as the scale domain, and create a default step range with 500ms per domain value. For example, the Gapminder domain is a list of every fifth year between 1955 and 2005, inclusive. The default range maps 1955 to 0ms, 1960 to 500ms, 1965 to 1000ms, and so on. A user can override this default range to slow down or speed up the animation.

Though the default domain is sufficient to express most common animations, a user may want to override the domain. Supplying a custom domain is useful for specifying non-keyframe-based animations that require direct reference to in-between values, or require animating through values that are missing from the dataset. For example, Fig. 3 shows an example of such a use case. The animation should advance



```
{
  "name": "open_dunks",
  "select": {
    "type": "point", "on": "timer",
    "predicate": {
      "and": [
        {"field": "open_datetime", "lte": {"expr": "anim_value"}},
        {"field": "closed_datetime", "gt": {"expr": "anim_value"}}
      ]
    }
  }
 "color": {
   "condition": {"param": "open_dunks", "value": "yellow"},
   "value": "grey"
 }
  "time": {
    "field": "open_datetime",
    "scale": {
      "type": "linear",
      "domain": [1632456000, 1632540600],
      "range": [0, 10000]
    }
  }
```

Fig. 3. Animation of Dunkin' Donuts stores' opening and closing times. With a custom domain and predicate, the animation advances through 24 hours at a constant rate and conditionally colors each store if the current time is between the store's open and close times.

through 24-hour time span at a constant rate. However, the dataset does not contain a field that has values that are evenly spaced in the desired domain. So, with a default scale domain, the animation would appear to jump between time stamps rather than move through them smoothly. To achieve the desired behavior, the user instead specifies a custom domain representing the continuous interval between 00:00 and 23:30.

4.1.3 Re-scale

By default, the visualization's data rectangle (or viewport) is fixed to the initial extents of the x- and y-scales calculated from the full dataset. However, for keyframe animations, only a subset of data is shown at any given time. If a user wants to re-calculate the viewport bounds based on only the data included in the current keyframe, rather than the original full dataset, they can set a flag in the time encoding called `rescale`. When `rescale` is `true`, the viewport's bounds are recomputed at each step of the animation. We refer to this concept as re-scaling because re-calcuating the viewport bounds involves updating the domains of the `x` and `y` scales at each keyframe.

Fig. 4 demonstrates the use of `rescale`. Rescale is enabled in Fig. 4A, where the viewport updates according to the current selection. The visualization remains tightly zoomed on the currently displayed bars, with the longest bar always scaled to nearly the full width of the viewport. In contrast, Fig. 4B has rescaling disabled. The viewport is initially calculated with the full dataset and remains fixed. This would be appropriate for Gapminder, because we want to show the countries moving along a fixed scale. However, it is less helpful for bar chart race. Instead of enabling positional comparisons to a fixed scale, the animation prioritizes making the ordering of the top-ranked bars salient.

4.2 Selections with a Timer Event Stream

Selections are subsets of data points that are populated when updates occur in an *event stream*. In Vega-Lite's interactive grammar, selections are defined using streams of user input events (e.g., clicks, mouse movements, or keyboard presses). The system uses the event's properties to query a set of data points. The selected data can then be applied to update downstream primitives in the visualization specification including data transformations, scale functions, or conditional visual encodings. For example, a selection defined using the `mouseover` event may be used to highlight marks that a user hovers over with their cursor. Under the hood, the selection receives a stream of `mouseover` events with `x` and `y` coordinates in pixels. It uses the scales associated with the `x` and `y` encoding channels to invert these screen coordinates back to data coordinates (i.e. values in the domain of the corresponding scale). A default predicate function iterates over all rows in the dataset, and includes the rows matching those data values in the selection.

Animated selections are analogous to interactive selections. However, instead of reacting to input events, animated selections use a `timer` event stream to advance an internal clock representing the elapsed time of the animation in milliseconds (ms). This clock resets to 0ms when it reaches the end of the range defined by the time encoding's scale (i.e. the animation loops the duration of the time scale's range). As the clock updates, the elapsed time value is mapped to a value in the time domain (i.e. the time encoding's field values). The animation selection updates to include all data points matching that value.

As selections rely on scales to convert map time to data values, selection-based animations still require a time encoding channel to be defined. In fact, all animations that can be expressed with only a time encoding can be elaborated into selection-based animations. In other words, selection-based animations are strictly more expressive than animations using only time encoding.

4.2.1 Applying Selections

In Vega-Lite, selections can be applied to other language constructs, including conditional mark encodings, scale domains, or data transformations [52]. This property of composition continues to hold with Animated Vega-Lite: animated and interactive selections can be used interchangeably wherever selections are supported in the Vega-Lite language. Therefore, selections driven by timer events inherit the expressiveness of interactive selections in terms of Yi et al.'s taxonomy of interaction techniques [51]. Animations can be used to: *select* marks of interest; *explore* subsets of data (panning and zooming); *reconfigure* data into different transformed states, *connect* related items; *abstract/elaborate* through overview and detail; and *filter* data dynamically. However, they cannot be used to change the properties of visual encodings on the fly, which is an interaction technique that falls outside of the selection-based model and is a limitation of base Vega-Lite.

4.2.2 Predicate

As the animation's elapsed time advances, the selection uses the scale defined in the time encoding to invert elapsed milliseconds (in the scale's range) to a data value (in the scale's domain). As a result, at any given time, there is an internal variable that has a data value corresponding to the animation's current time. When the Vega-Lite specification is compiled into Vega, this variable is represented as a Vega signal called `anim_value`. In the Gapminder example, `anim_value` starts at `1955` at 0ms, and advances to `1960, 1965, ..., 2005`.

To construct keyframes, the selection queries a subset of data tuples to include in the keyframe based on the current value of `anim_value`. By default, tuples are included in the keyframe if their value in the time encoding's field (e.g. `year` for Gapminder) is equal to `anim_value`. However, to define alternate inclusion criteria for determining keyframes, users can specify custom predicate functions. For example, if at every step of the animation, a user wished to show all points with year less than or equal to `anim_value`, they would use the following predicate:

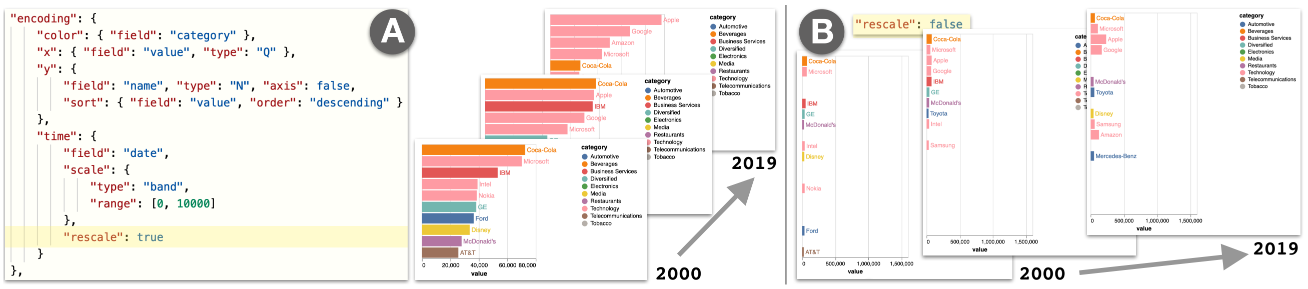
```
{"field": "year", "lte": "anim_value"}
```

Previously, Vega-Lite did not allow users to customize the selection predicate because the majority of interactions could be expressed using a combination of default predicates and selection transformations. Nonetheless, enabling predicate customization in the selection specification also increases the expressiveness of the interactive grammar.

4.2.3 Input Element Binding

Using the `bind` property, a user can populate a selection using a dynamic query widget (such as an HTML slider or checkbox). For animated selections, input element binding offers a convenient way to add interactive playback control to the animation. For instance, the user can bind an animated selection to a checkbox to toggle whether the animation is playing or paused. Similarly, they can bind a selection to a range slider and drag to scrub to a specific time in the animation.

Scrubbing the animation with the slider surfaces an interesting design challenge when combining animation and interaction: how should the system delegate control between the animation timer and user interaction? Initially, the animation is driven by the timer, with the slider visualizing timer updates. When the user starts dragging the slider, the system pauses the animation and delegates control to user interaction. Pausing is necessary so that the slider does not continue to advance forward while the user is currently scrubbing. When the user is done scrubbing, they may want to give control back to the animation. To


```
"encoding": {
    "color": { "field": "category" },
    "x": { "field": "value", "type": "Q" },
    "y": {
        "field": "name", "type": "N", "axis": false,
        "sort": { "field": "value", "order": "descending" }
    },
    "time": {
        "field": "date",
        "scale": {
            "type": "band",
            "range": [0, 10000]
        },
        "rescale": true
    }
},
```

```
"rescale": false
```

Fig. 4. Demonstration of the *rescale* time encoding property recreating a D3 bar chart race example [28]. (A) `rescale` is `true`: the viewport is recalculated on each keyframe. (B) `rescale` is `false`: the viewport is calculated on the whole dataset, and does not update with the selection.

facilitate this, Animated Vega-Lite automatically includes a play/pause checkbox alongside bound sliders. The user can simply re-check the box to give control over the animation back to the timer.

4.2.4 Pausing

Animated Vega-Lite supports pausing in two ways: by interaction, and by data value. Interactive pauses are specified using the `filter` property of Vega-Lite event streams. Users can provide the name of a Vega-Lite parameter to the `filter` property of a timer event stream. Parameters can be either selections or variables. When the provided parameter evaluates to true (i.e. is a non-empty selection or a true boolean variable), the filter will capture incoming events, preventing the animation clock from advancing. When the paramater evaluates to false, the events will resume propagating and the animation will continue. For example, a user can bind a checkbox to a parameter named `is_playing`, and use the following event stream definition to pause the visualization when the box is checked:

```
"on":{"type": "timer", "filter": "is_playing"}
```

Pausing by data value is specified using the `pause` property of an animated selection definition. The user provides a list of data values to pause on, and the duration of each pause. For example, a user can specify that the Gapminder animation should pause on the year 1995 for 2 seconds, to draw attention to the data for that year:

```
"pause": [{"value": 1995, "duration": 2000}]
```

4.2.5 Global Easing

Easing is a common animation technique that involves controlling the rate that the animation timer advances. Easing is typically implemented using a palette of pre-defined functions that map an animation time domain to a transformed time domain. For example, an exponential easing function might cause the animation clock to begin advancing slowly, and then exponentially accelerate as the animation progresses. In Animated Vega-Lite, the animation clock advances linearly by default. However, users can use the `easing` property of a selection to specify an easing function to apply to the whole duration of the animation. Animated Vega-Lite exposes D3's named easing functions [27].

5 IMPLEMENTATION

We implement Animated Vega-Lite using a prototype compiler, wrapping the existing Vega-Lite compiler to ingest Animated Vega-Lite specifications and output a lower-level Vega specification. The Animated Vega-Lite prototype compiler begins by expanding a user-supplied specification into a "normalized" format with all implicit default values filled in explicitly. This step includes generating default selections and transforms for animations specified using only `time` encodings, and filling in default scale and key definitions. This normalized specification is passed to the next compiler step to simplify processing.

To convert Animated Vega-Lite into low-level Vega, we use the existing Vega-Lite compiler to make the initial conversion into Vega (using a copy of the specification with animation removed), and then call a series of functions to compile animation-specific parts of the spec and merge them with the output Vega. Because Vega-Lite's high-level abstractions do not have a one-to-one mapping to low-level Vega concepts, seemingly-isolated Vega-Lite fragments will typically make changes in many different parts of the Vega spec. Each of these functions takes in fragments of Animated Vega-Lite and standard Vega, and outputs a partial Vega specification that includes dataset, signal, scale, and mark definitions to merge into the output.

Compilation happens in six steps. First, `compileAnimationClock` uses definitions of animated selections and time encoding channels to create Vega signals and datasets for controlling the current state of the animation, handling pausing, and interfacing with interactive playback controls. Next, `compileTimeScale` takes in a definition of a time encoding alongside Vega marks and scales. It creates Vega-level scales for the time encoding, and signals to handle inversions between the animation clock and the corresponding data value at that time. It also applies rescaling to mark encodings if applicable. `compileAnimationSelections` then ingests definitions of animated selections to produce Vega signals and datasets that implement custom predicates, pausing and easing, and input element binding. Fourth, `compileFilterTransforms` takes animation selections and any filter transforms that reference those selections, and materializes the selections as filtered datasets in Vega. These datasets provide the backing data for rendering marks at each keyframe. `compileKey` then uses the time encoding specification to generate datasets and signals that handle tweening between keyframes. Finally, `compileEnterExit` supports top-level enter and exit encoding definitions in Animated Vega-Lite, converting them into Vega-level enter and exit encodings. Because of existing limitations in Vega, enter and exit currently are not well-supported for animation. However, pending Vega support, designers should be able to control the behavior of visual encodings as marks enter and exit the current keyframe.

We chose to implement our compiler as a wrapper around the existing Vega-Lite compiler in order to facilitate rapid prototyping. However, our current approach faces performance challenges that could be improved with internal changes to Vega and Vega-Lite. For example, we currently support tweening by creating three separate datasets: the current keyframe, the next keyframe, and a joined dataset with tweens computed as a derived column. This expensive operation causes noticeable lag on large datasets. In future implementations, we can instead create a Vega dataflow operator that leverages the animation's semantics to compute tweens more efficiently. For example, instead of computing multiple datasets independently and performing a join, the operator can create a single dataset backed by a sliding window over the time facets.

6 EVALUATION: EXAMPLE GALLERY

To evaluate Animated Vega-Lite's expressiveness, we created an example gallery to demonstrate coverage over both Yi et al.'s taxonomy of interaction intents [51] and Heer & Robertson's taxonomy of transition types in animated statistical graphics [12]. As Fig. 5 shows, we support 6 / 7 interaction categories and 5 / 7 animation categories.

Fig. 5a demonstrates an overview + detail visualization. A selection controls a brush over the bottom view, which sets the zoomed viewport of the top view. This selection is defined using a predicate that defines a sliding window over the x-axis field. When the brush is driven by animation, the selection is updated on each timer event. When the brush is driven by interaction, the selection is instead updated on drag events. Because the original Vega-Lite selection model unifies panning and zooming as selections applied to a scale domain, this approach can be adapted to animate arbitrary geometric panning and

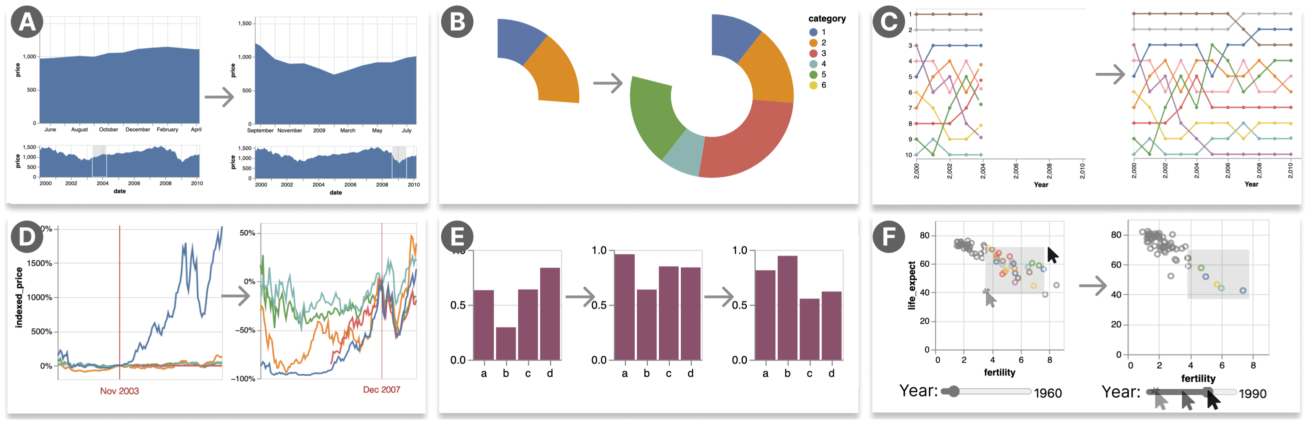


Fig. 5. Animated Vega-Lite examples demonstrating coverage over interaction and animation taxonomies [12,51] (see Fig. 4 for an example *substrate transform* and Fig. 3 for *select*). A) *View transform* via panning, *abstract/elaborate* via overview + detail, and *connect*ing multiple views. B) *Filtering* data via a predicate. C) *Ordering* / *reconfiguring* a sorted axis in a bump chart. D) *Exploring* sequential *timestep*s of an index chart. E) A hypothetical outcome plot in the style of the New York Times [15]. F) An interactive brush selection over Gapminder.

zooming behavior. This visualization demonstrates a *view transformation*, changing the reader's viewpoint by panning and zooming the top view. It also demonstrates an *abstract/elaborate* intent by showing the data at different levels of detail in the top and bottom view, and the *connect* intent by showing corresponding data across multiple views.

Fig. 4 shows a bar chart's x-scale dynamically recalculating on each frame using the `rescale` property of a time encoding (Sect. 4.1.3). This animation technique demonstrates a *substrate transformation* through scale manipulations. It also demonstrates the *reconfigure* intent by showing a new spatial arrangement of the data.

In Fig. 3 and Fig. 5b, we apply a conditional filter over the whole dataset, with filter parameters changing over time. In contrast to faceting, filtering can leverage custom selection predicates to show and hide data — a single data point can appear in multiple groups. Both taxonomies contain a category for *filter*ing, shown here by adding or removing elements from the display. Fig. 3 additionally demonstrates a *select* intent by using conditional encoding to highlight selected data.

Fig. 4 and Fig. 5c show examples with a sorted axis. When a `key` is specified in a time encoding, the system automatically tweens an element's position even when its sort index has changed in the next keyframe. Continually sorting elements as the underlying data changes demonstrates an *or   ing* transition, as well as a *reconfigure* intent.

Time encodings transition between sequential time values by default in Animated Vega-Lite (e.g. Fig. 2). Fig. 5d demonstrates an additional example of this animation. A default animated point selection is applied to a data transform that re-normalizes a stock price time-series chart on each tick. The original Vega-Lite paper contains an interactive version of this example, which instead populates the point selection on mouse hover events [36]. These examples demonstrate *timestep* transitions, which also fulfill the *explore* intent by showing new data points at each step. Axis re-normalization is also an example of a *reconfigure* intent.

In addition to achieving broad coverage over the two taxonomies, our system also supports simulation techniques including hypothetical outcome plots (Fig. 5e) [13]. And, as previously discussed in Sect. 4.2.1, animated selections can be applied to the same set of dynamic visual behaviors as interactive selections. Consequently, users can easily switch between timer and input event streams when prototyping existing interaction techniques in Vega-Lite. For example, Fig. 5a and Fig. 5d show animated selections driving common interaction techniques — panning and re-normalizing, respectively. Users can also easily compose interaction techniques with animated visualizations by defining additional selections. For example, Fig. 5f demonstrates an interactive brush used to highlight a region of an animated Gapminder visualization. Points of interest are conditionally colored as they enter or exit the brush region.

Discussion and Limitations. Like the original Vega-Lite, Animated Vega-Lite intentionally trades some limits to expressivity for gains in concise, high-level, declarative specification. In Sects. 7.2.1 & 7.2.2, we detail this expressiveness tradeoff in terms of the classes of animation techniques (Animated Vega-Lite primarily supports *scene* techniques instead of *segue*) as well as the implications on how keyframes are modeled and generated (Animated Vega-Lite supports non-parametric keyframe transitions, and offers some limited support for parametric keyframe transitions). Thus, lower-level and imperative languages will necessarily be more expressive: for instance, D3 can express both scene and segue animations, but using different language constructs (timer event loops and transition functions, respectively). As these sections describe, offering high-level declarative specification that unifies not only these distinct conceptual models of animation, but also interaction and static charts, remains a compelling direction for future work.

By extending Vega-Lite, Animated Vega-Lite also inherits its predecessor's limitations. For instance, Vega-Lite selections cannot alter visual encodings or data transformation pipelines at runtime (the *encode* interaction type in Yi et al.'s taxonomy [51]); thus, Animated Vega-Lite cannot support the *visualization change* or *data schema change* transition types in the Heer & Robertson taxonomy [12].

7 Evaluation: Critical Reflection

To identify our grammar's design tradeoffs, we compared our approach to existing animated visualization grammars following the *critical reflections* evaluation method [35]. We recruited five developers of existing grammars: John Thompson and Leo Zhicheng Liu[1] of Data Animator [46], Tong Ge of Canis [10] and CAST [9], Thomas Lin Pedersen of gganimate [43], and Younghoon Kim of Gemini [19] and Gemini[2] [20]. We focused on animation grammar developers because the interactive grammar was evaluated in the original Vega-Lite paper. With each participant, we conducted a one-hour pre-interview. We then asked them to asynchronously engage with our grammar for an extended time by reading a system walkthrough and grammar documentation similar to Sect. 3 and Sect. 4, respectively, and run examples similar to those found in Sect. 6. We further suggested participants write new specifications and/or port other examples, including examples from their own tools. We encouraged participants to take notes and reflect on the design of Animated Vega-Lite during the process. Finally, we conducted post-interviews with each participant that lasted 30–60 minutes. Each participant was offered a $125 gift card as compensation.

Our goals were to (i) compare and contrast their design processes with ours, (ii) understand differences and design tradeoffs between their grammars and ours, and (iii) generate insights about the direction of future animation grammars. During the interviews, three of the authors of this paper began developing initial thematic hypotheses. After the interviews, we independently conducted a thematic analysis before finally coming together and synthesizing our insights, which we summarize below. These themes provide insight into the design of our grammar, and animated visualization grammars more generally.

[1]Thompson & Liu also co-authored the original critical reflections paper [35].

7.1 Grammar Design Process

7.1.1 Specific Examples Motivate Grammar Design

When scoping their research projects, our interviewees prioritized motivating examples that they found personally compelling. For example, the authors of Data Animator and Gemini were both motivated in part by R2D3 [40]. As we discuss in the following subsections, the choosing examples to support leads to design tradeoffs, e.g. between scene- and segue-dominant abstractions (Sect. 7.2.1). Thus, a handful of compelling in-the-wild examples can significantly influence the grammars developers build. Other examples that were cited across multiple interviews included Gapminder [34], Periscopic's Gun Deaths [31], and animations in the New York Times (NYT) and the Guardian.

On the other hand, a *lack* of existing examples may also motivate a grammar developer. For example, to gain more insight into the popularity of animated visualization techniques, Kim scraped NYT and Guardian articles from 2018 as well as YouTube videos from the same year. He noticed that about 90% of the animated visualizations he studied updated data, but kept the encoding fixed. R2D3 was a notable exception. A similar imbalance can be found in the Data-Gifs example gallery [39], where over half of the examples have fixed encodings. Kim hypothesized that the imbalance is influenced by the affordances of existing tools, and decided to optimize Gemini for transitions between changing encodings.

With Animated Vega-Lite, we were motivated by the large collection of existing examples with static encodings, such as those in the Data-Gifs example gallery. This category includes many prominent designs like Gapminder and bar chart races. Rather than focus on developing an expressive language of transitions between keyframes, we focused on an expressive language of keyframe generation via selections. Our abstractions facilitate the design of visualizations that must produce many keyframes backed by a fixed encoding.

7.1.2 Natural Programming vs. Core Calculus Design

To make their systems easy to use for their target audiences, the authors of Data Animator and Gemini aimed to develop grammars that matched the existing mental models of animation designers. To that end, both groups conducted interviews prompting experienced animators to sketch interfaces or write pseudocode to recreate exemplar animated visualizations [19,45]. Fundamental abstractions emerged from these formative studies. For instance, Gemini's studies yielded the concepts of synchronizing (*'at the same time'*) and concatenating (*'then'*, *'after'*) while Data Animator's studies surfaced designers' familiarity with keyframes in Adobe After Effects. This design process is known as *natural programming*, where a developer aims *"for the language and environment to work the way that nonprogrammers expect"* [30].

In contrast, we set out to develop a small *core calculus* [6] of abstractions for Animated Vega-Lite, which we outlined in Sect. 4. Our design was motivated by the desire to explore whether interaction and animation could be unified. This unification would likely not have been elicited by a target user. Because the key idea of our paper is to identify a unified abstraction, this difference in approach results in a design tradeoff. As Kim explained, Animated Vega-Lite may seem natural to a Vega-Lite user, but might present a steeper learning curve to someone familiar with animation tools like Adobe AfterEffects, as Animated Vega-Lite has no explicit concept of a keyframe.

Analyzing these processes via the Cognitive Dimensions of Notation [44], we find that iterating closely with end users in a natural programming process yields a grammar that *closely maps* to common user mental models. On the other hand, by distilling abstractions to a reduced set of orthogonal concepts, a core calculus process better emphasizes a *consistent* API that has low *viscosity*. Over-emphasizing one process or the other may drag a language design too far to one side. With PLIERS, Coblenz et al. [6] offer suggestions for how developers may integrate and balance between these approaches. They recommend a developer iterate between developing the theoretical foundations of their language (core calculus) and the user-facing language (surface language). Moreover, Coblenz et al. suggest adapting natural programming by *progressively prompting* a user with incrementally more information about a language's proposed API. This additional scaffolding can help scope how natural programming studies explore mental models, and also lets a language developer gain insights even when the core calculus significantly departs from a user's familiar models. Integrated design processes, like PLIERS, are likely to be valuable methods for assessing future unified grammars, because these systems must balance significant conceptual unifications with end-users' ease-of-use.

7.2 Animation Abstractions and Design Considerations

7.2.1 Scene- vs. Segue-Dominant Abstractions

Several interviewees noted that Animated Vega-Lite's abstractions appear complementary to their systems. For example, Kim noted his conceptual distinction between Animated Vega-Lite and Gemini is *"[Animated Vega-Lite] animates the internal state within Vega-Lite, and Gemini doesn't care about the internal state. It just transforms between two static states of Vega-Lite."* Similarly, Thompson said *"if you compare [Animated Vega-Lite] directly to Data Animator, the two of them together would be really nice. What one doesn't have, the other does really well."* For instance, he highlighted Animated Vega-Lite's ability to automatically generate keyframes from data (e.g., each `year` keyframe in Gapminder) and Data Animator's ability to precisely specify transitions between keyframes (such as staggering) as complementary components of the two systems. He also appreciated Animated Vega-Lite's ability to create overlapping keyframes via layering, as in our bar chart race example (Fig. 4). Pedersen provides one explanation for why our approach is complementary to the existing systems we studied. In his useR! 2018 keynote, Pedersen introduced the concepts of a *scene* and a *segue* animation [41]. A scene animation, such as Gapminder, is one where the data is changing (such as countries ranging over years), but the visual encoding is not. One can imagine a scene playing within a fixed stage (i.e., a static visual encoding). In contrast, a segue animation — such as a pie chart transitioning to a bar chart — is one where the visual encoding is changing, but the data is fixed. In practice, the line between a scene and segue is not always clear. For example, transitioning from a strip plot to a box and whiskers plot involves both a change to the data (computing aggregate quantities) and a change to the visual encoding (converting to box-and-whiskers).

Using this scene and segue distinction, Animated Vega-Lite and gganimate may be categorized as *scene-dominant* grammars. Both systems aim to cover a large space of animated visualizations with fixed encodings, such as Gapminder and bird migrations. Both systems support an additional collection of visual encoding transformations. For example, Animated Vega-Lite supports rescaling, panning, and zooming while gganimate supports transitions that can interpolate between different shapes with the same underlying data. Though both Animated Vega-Lite and gganimate are scene-dominant systems, Pedersen highlighted the expressiveness of Animated Vega-Lite's selection model for generating arbitrary keyframes from data (as shown with the Dunkin example in Fig. 3) as a key conceptual distinction between the two.

On the other hand, Data Animator, Canis, and Gemini are *segue-dominant*. These systems have focused primarily on connecting two distinct keyframes that may have distinct visual encodings and data. To construct a transition, Data Animator, Canis, and Gemini each construct a mapping between two keyframes. This approach works well when the data set is fixed, and there are only a few keyframes (as is typical when showing a small handful of segues). But as identified by Thompson and Liu, to support an animation like Gapminder, these systems must produce a keyframe for every year in the dataset.

As discussed in Sect. 6, Animated Vega-Lite inherits Vega-Lite's inability to represent complex runtime changes to visual encodings and data transformations. We suspect that extending Vega-Lite with these capabilities could enable segue animations in a future version of Animated Vega-Lite. To support complex runtime changes, Vega-Lite's conditional encodings could be extended from just mark properties to mark types and data transforms as in Ivy [25]. And our support for enter and exit could be extended to operate not just on data, but also on these more expressive encoding changes.

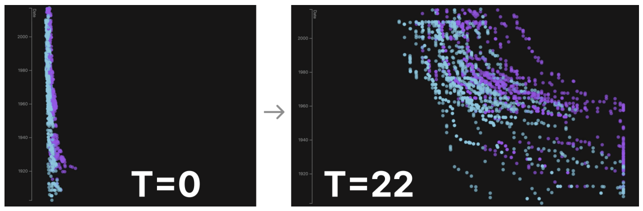


Fig. 6. Swimming World Records example from Data Animator [16].

7.2.2 Modeling Transitions Between Keyframes

Keyframes were the most salient animation abstraction in our interviews. We discussed keyframe concepts with every interviewee, and they would often use keyframes to pose comparisons between different systems' abstractions. Every tool had to make decisions about (i) how to generate keyframes and (ii) how to transition between them. Moreover, keyframes and transitions are useful abstractions for both scene- and segue-dominant systems. In this subsection we surface an axis of the keyframe design space: modeling transitions between keyframes.

Non-parametric transitions. The simplest kind of transition between keyframes is a non-parametric transition. Consider a linear sequence of keyframes, where each keyframe describes an entire scene-graph. Transitions between these keyframes are *non-parametric* in that the same transition is applied to every data point. For example, changing every bar to a point in 0.5 seconds (a segue animation) is a non-parametric transition because the transition's definition is independent of the mark's encoded data — i.e. its duration is a constant value. Similarly, animating countries in Gapminder (a scene animation) is also a non-parametric transition because the transition applied to each mark is identical (moving between two points in a fixed time interval).

Animated Vega-Lite supports non-parametric transitions via its timer, easing, and interpolation abstractions, which implicitly specify a transition across keyframes. The other libraries also support non-parametric transitions between pairs of keyframes, but only scene-dominant systems (gganimate and Animated Vega-Lite) support non-parametric transitions across *many* keyframes. In scene-dominant animations, the same transition specification can be reused across a sequence of keyframes sharing a fixed encoding.

Parametric transitions. In contrast to non-parametric transitions, *parametric* transitions involve transition definitions that depend on the backing data. A common use case for this model is to stagger transitions — a common segue technique that applies a small delay to each animated element to make them easier to track [12]. Because parametric transitions depend on data, individual marks can have different timing properties during the same transition.

Segue-dominant systems Data Animator, Canis, and Gemini all support parametric transitions. But, as Thompson identified in his post-interview, parametric transitions also increase the expressive gamut of scene animations. For example, Fig. 6 shows "Swimming World Records Throughout History" from the Data Animator example gallery. This animated scatterplot shows replays of world record swimmers. The input data includes swimmers and their final race times. When Thompson tried to port this example to Animated Vega-Lite, he realized he *"had no clue how to do it. The two keyframes in this example are very simple. All of the circles at one x position, and then all of the circles like 200–400 pixels to the right. For us, you change the* speed *of each individual shape based on a data property."* Animated Vega-Lite could support this animation by allowing users to explicitly define a transition, with its speed parameterized by a data value.

To support parametric transitions, future versions of Animated Vega-Lite could use Lu et al.'s concept of "dynamic functions" [24]. These functions use mappings between data and transitions to specify rate-of-change properties of transitions over time (e.g., encoding transition speed instead of mark position). Adapting this segue-dominant concept to Animated Vega-Lite could increase expressivity, though further work is required to understand its composition with and implications for static and interactive language constructs. For instance, segue transition properties may more easily compose with existing static and interactive Vega-Lite constructs if translated back into scene keyframes as direct encodings instead of rates (e.g. instantiating transition speed as additional position keyframes). However, this would trade off the memory efficiency of the segue representation.

Connecting transitions in series and parallel. Some of the most compelling animated examples cannot be represented as a linear sequence of transitions, parametric or not. For instance, Periscopic's Gun Deaths animation [31], a visualization frequently cited by our interviewees, cannot easily be represented even by parametric transitions. When discussing this example, Thompson remarked: *"This was one that I had on my list of* 'oh it would be so cool if we could create this,' *and then I could just not figure out a way of doing it. [...] How do you have the circle appear and then drop, and then the line keeps going? I have no clue how to do that [in Data Animator]"*. Authoring this animation is difficult because there is no linear transition specification: the animation splits in two when the circle drops and the line continues. We are not certain that *any* of the grammars we have discussed in our critical reflections can easily express this animation, because it involves both scene and segue animation.

Gemini's *composition rules* offer a promising path for the transitions necessary to support the Gun Deaths animation. Gemini's *concat* primitive allows a user to specify animations in series, while its *sync* primitive allows a user to specify animation components that play in parallel. Using these primitives, one could specify a sync that splits the animation into the circle and the line, and then concat the many stages of the Gun Deaths animation together. More generally, concat and sync allow a user to model transitions as a *series-parallel* graph [48].

However, this abstraction alone is not enough. While Gemini has a rich transition language, it cannot generate keyframes automatically from data like Animated Vega-Lite. This generation is necessary for the Gun Deaths animation to visualize individual points. Combining Gemini's segue abstractions with Animated Vega-Lite's scene abstractions is a promising future direction for expressive animation.

8 Conclusion and Future Work

Animated Vega-Lite contributes a low viscosity, compositional, and systematically enumerable grammar that unifies specification of static, interactive, and animated visualizations. Within a single grammar, authors can now easily switch between the three modalities during rapid prototyping, and also compose them together to effectively communicate and analyze faceted and time-varying data.

Our grammar takes a promising step in helping authors develop visualizations that leverage the dynamic affordances of computational media. During interviews, Pedersen described unification as the "holy grail" of data visualization APIs: *"A grammar of graphics that defines how things look, a grammar of animation that defines how things react, and a grammar of interaction that defines how things interact. Having all of that in one unified theoretical framework would simply be awesome."* Future work might more deeply explore the distinctions and tradeoffs we surfaced between transition and keyframe models, or study the implications of unification at the lower-level of reactive programming semantics and data stream management.

Beyond language design, we hope that Animated Vega-Lite facilitates future work on interactive and animated visualization akin to the role the original Vega-Lite has played. For instance, how might we leverage Animated Vega-Lite's ability to enumerate static, interactive, and animated visualizations to study how these modalities facilitate data analysis and communication — replicating and extending prior work [33] more systematically? Similarly, how might study results be codified in the Draco knowledge base [29], or exposed in systems like Voyager [49, 50] or Lux [23] to recommend animated visualizations during exploratory data analysis? To support this future research, we intend to contribute our work back to the open source Vega-Lite project.

Acknowledgments

We thank our critical reflections interlocutors and anonymous reviewers. This work was supported by NSF grants #1942659 and #1900991 and by the NSF's SaTC Program. This material is based upon work supported by the National Science Foundation under Grant No. 1745302.

REFERENCES

[1] Plotly Graphing Libraries, 2012. https://plotly.com/graphing-libraries/.

[2] F. A. Abukhodair, B. E. Riecke, H. I. Erhan, and C. D. Shaw. Does interactive animation control improve exploratory data analysis of animated trend visualization? In *Visualization and Data Analysis 2013*, vol. 8654, pp. 211–223. SPIE, Feb. 2013. doi: 10.1117/12.2001874

[3] M. Bostock, V. Ogievetsky, and J. Heer. D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185

[4] H.-J. Bucher and P. Schumacher. The relevance of attention for selecting news content. An eye-tracking study on attention patterns in the reception of print and online media. *Communications*, 31(3), Jan. 2006. doi: 10.1515/COMMUN.2006.022

[5] F. Chevalier, P. Dragicevic, and S. Franconeri. The Not-so-Staggering Effect of Staggered Animated Transitions on Visual Tracking. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2241–2250, Dec. 2014. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2014.2346424

[6] M. Coblenz, G. Kambhatla, P. Koronkevich, J. L. Wise, C. Barnaby, J. Sunshine, J. Aldrich, and B. A. Myers. PLIERS: A Process that Integrates User-Centered Methods into Programming Language Design. *ACM Transactions on Computer-Human Interaction*, 28(4):28:1–28:53, July 2021. doi: 10.1145/3452379

[7] P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete. Temporal distortion for animated transitions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2009–2018. ACM, Vancouver BC Canada, May 2011. doi: 10.1145/1978942.1979233

[8] C. Elliott and P. Hudak. Functional reactive animation. In *Proceedings of the second ACM SIGPLAN international conference on Functional programming*, ICFP '97, pp. 263–273. Association for Computing Machinery, New York, NY, USA, Aug. 1997. doi: 10.1145/258948.258973

[9] T. Ge, B. Lee, and Y. Wang. CAST: Authoring Data-Driven Chart Animations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pp. 1–15. Association for Computing Machinery, New York, NY, USA, May 2021. doi: 10.1145/3411764.3445452

[10] 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*, 2020. Publisher: The Eurographics Association and John Wiley & Sons Ltd. doi: 10.1111/cgf.14005

[11] E. Greussing, S. H. Kessler, and H. G. Boomgaarden. Learning From Science News via Interactive and Animated Data Visualizations: An Investigation Combining Eye Tracking, Online Survey, and Cued Retrospective Reporting. *Science Communication*, 42(6):803–828, Dec. 2020. Publisher: SAGE Publications Inc. doi: 10.1177/1075547020962100

[12] J. Heer and G. Robertson. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, Nov. 2007. doi: 10.1109/TVCG.2007.70539

[13] J. Hullman, P. Resnick, and E. Adar. Hypothetical Outcome Plots Outperform Error Bars and Violin Plots for Inferences about Reliability of Variable Ordering. *PLOS ONE*, 10(11):e0142444, Nov. 2015. Publisher: Public Library of Science. doi: 10.1371/journal.pone.0142444

[14] J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, 9(3):90–95, May 2007. Conference Name: Computing in Science Engineering. doi: 10.1109/MCSE.2007.55

[15] N. Irwin and K. Quealy. How Not to Be Misled by the Jobs Report. *The New York Times*, May 2014.

[16] John Thompson. Swimming World Records throughout History, 2020.

[17] A. Kale, F. Nguyen, M. Kay, and J. Hullman. Hypothetical Outcome Plots Help Untrained Observers Judge Trends in Ambiguous Data. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):892–902, Jan. 2019. doi: 10.1109/TVCG.2018.2864909

[18] Y. Kim, M. Correll, and J. Heer. Designing Animated Transitions to Convey Aggregate Operations. *Computer Graphics Forum*, 38(3):541–551, 2019. doi: 10.1111/cgf.13709

[19] Y. Kim and J. Heer. Gemini: A Grammar and Recommender System for Animated Transitions in Statistical Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):485–494, 2021. doi: 10.1109/TVCG.2020.3030360

[20] Y. Kim and J. Heer. Gemini^2: Generating Keyframe-Oriented Animated Transitions Between Statistical Graphics. In *2021 IEEE Visualization Conference (VIS)*, pp. 201–205. IEEE, New Orleans, LA, USA, Oct. 2021. doi: 10.1109/VIS49827.2021.9623291

[21] B. Kondo and C. Collins. DimpVis: Exploring Time-varying Information Visualizations by Direct Manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2003–2012, Dec. 2014. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2014.2346250

[22] F. A. La Sorte, D. Fink, W. M. Hochachka, and S. Kelling. Convergence of broad-scale migration strategies in terrestrial birds. *Proceedings of the Royal Society B: Biological Sciences*, 283(1823):20152588, Jan. 2016. Publisher: Royal Society. doi: 10.1098/rspb.2015.2588

[23] D. J.-L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, and A. G. Parameswaran. Lux: always-on visualization recommendations for exploratory dataframe workflows. *Proceedings of the VLDB Endowment*, 15(3):727–738, Nov. 2021. doi: 10.14778/3494124.3494151

[24] M. Lu, N. Fish, S. Wang, J. Lanir, D. Cohen-Or, and H. Huang. Enhancing Static Charts With Data-Driven Animations. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2628–2640, July 2022. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2020.3037300

[25] A. M. McNutt and R. Chugh. Integrated Visualization Editing via Parameterized Declarative Templates. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–14. ACM, Yokohama Japan, May 2021. doi: 10.1145/3411764.3445356

[26] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. Flapjax: a programming language for Ajax applications. In *Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications*, OOPSLA '09, pp. 1–20. Association for Computing Machinery, New York, NY, USA, Oct. 2009. doi: 10.1145/1640089.1640091

[27] Mike Bostock. d3-ease, 2015. https://github.com/d3/d3-ease.

[28] Mike Bostock. Bar Chart Race, Explained, 2019. https://observablehq.com/@d3/bar-chart-race-explained.

[29] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, Jan. 2019. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2018.2865240

[30] B. A. Myers, J. F. Pane, and A. J. Ko. Natural programming languages and environments. *Communications of the ACM*, 47(9):47–52, Sept. 2004. doi: 10.1145/1015864.1015888

[31] Periscopic. United States gun death data visualization, 2013.

[32] 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)*, pp. 86–92, Oct. 2018. doi: 10.1109/BELIV.2018.8634297

[33] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of Animation in Trend Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1325–1332, Nov. 2008. doi: 10.1109/TVCG.2008.125

[34] H. Rosling. The best stats you've ever seen, 2006. https://www.ted.com/talks/hans_rosling_the_best_stats_you_ve_ever_seen.

[35] 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 on Visualization and Computer Graphics*, pp. 1–1, 2019. doi: 10.1109/TVCG.2019.2934281

[36] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, Jan. 2017. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2016.2599030

[37] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, Jan. 2016. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2015.2467091

[38] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, UIST '14, pp. 669–678. Association for Computing Machinery, New York, NY, USA, Oct. 2014. doi: 10.1145/2642918.2647360

[39] X. Shu, A. Wu, J. Tang, B. Bach, Y. Wu, and H. Qu. What Makes

a Data-GIF Understandable? *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1492–1502, Feb. 2021. doi: 10.1109/TVCG.2020.3030396

[40] Stephanie Yee and Tony Chu. A visual introduction to machine learning, Part II, 2015. http://www.r2d3.us/visual-intro-to-machine-learning-part-2/.

[41] Thomas Lin Pedersen. The Grammar of Animation, July 2018. https://www.youtube.com/watch?v=21ZWDrTukEs.

[42] Thomas Lin Pedersen. gganimate has transitioned to a state of release, 2019. https://www.data-imaginist.com/2019/gganimate-has-transitioned-to-a-state-of-release/.

[43] Thomas Lin Pedersen and David Robinson. A Grammar of Animated Graphics, 2019. https://gganimate.com/.

[44] Thomas RG Green. Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay, eds., *People and Computers V*, pp. 443–460. Cambridge University Press, Cambridge, UK, 1989.

[45] J. Thompson, Z. Liu, W. Li, and J. Stasko. Understanding the Design Space and Authoring Paradigms for Animated Data Graphics. *Computer Graphics Forum*, 39(3):207–218, 2020. doi: 10.1111/cgf.13974

[46] J. R. Thompson, Z. Liu, and J. Stasko. Data Animator: Authoring Expressive Animated Data Graphics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pp. 1–18. Association for Computing Machinery, New York, NY, USA, May 2021. doi: 10.1145/3411764.3445747

[47] B. Tversky, J. B. Morrison, and M. Betrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, Oct. 2002. doi: 10.1006/ijhc.2002.1017

[48] Wikipedia contributors. Series–parallel graph — Wikipedia, The Free Encyclopedia, 2022.

[49] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, Jan. 2016. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2015.2467191

[50] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 2648–2659. Association for Computing Machinery, New York, NY, USA, May 2017. doi: 10.1145/3025453.3025768

[51] J. S. Yi, Y. a. Kang, J. Stasko, and J. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, Nov. 2007. doi: 10.1109/TVCG.2007.70515

[52] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing Interactive Visualizations by Demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):304–314, Feb. 2021. doi: 10.1109/TVCG.2020.3030367