Deimos: A Grammar of Dynamic Embodied Immersive Visualisation Morphs and Transitions

Benjamin Lee
Benjamin.Lee@visus.uni-stuttgart.de
University of Stuttgart
Stuttgart, Germany
Monash University
Melbourne, Australia

Arvind Satyanarayan
arvindsatya@mit.edu
MIT CSAIL
Cambridge, United States

Maxime Cordeil
m.cordeil@uq.edu.au
University of Queensland
Brisbane, Australia

Arnaud Prouzeau
arnaud.prouzeau@inria.fr
Inria & LaBRI (University of Bordeaux, CNRS, Bordeaux-INP)
Bordeaux, France

Bernhard Jenny
bernie.jenny@monash.edu
Monash University
Melbourne, Australia

Tim Dwyer
tim.dwyer@monash.edu
Monash University
Melbourne, Australia

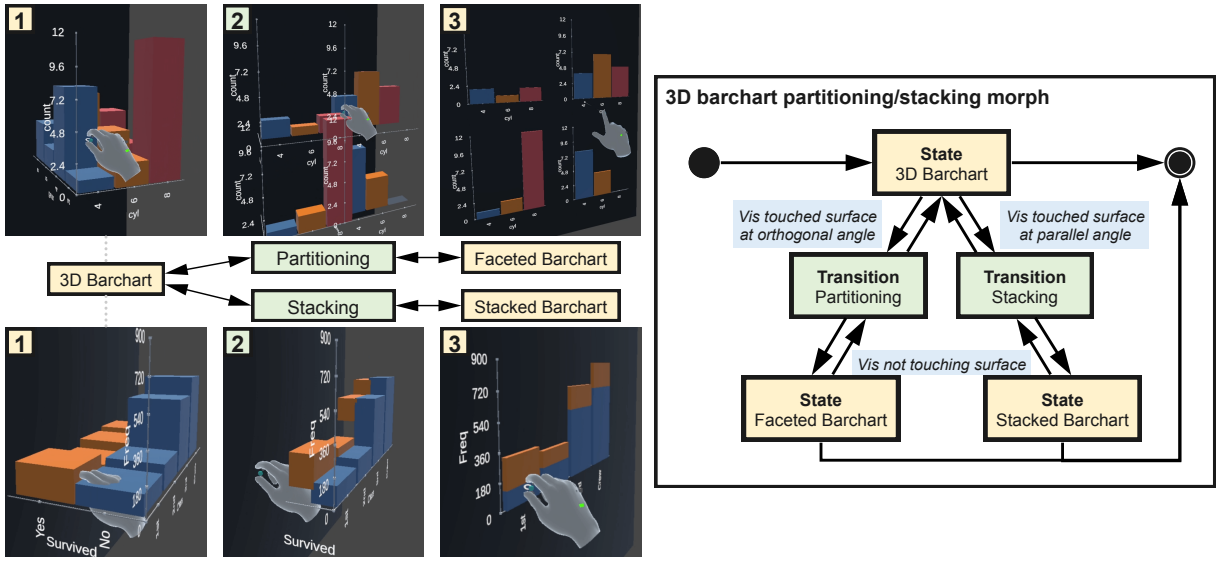

Figure 1: A morph that transforms any 3D barchart into either a faceted barchart or a stacked barchart when it touches a surface in the immersive environment, depending on its angle of intersection. Left: Still images of the animated transition. Images labelled "1" and "3" correspond to *states* (keyframes) in the morph, images labelled "2" correspond to the *transitions* in the morph. Right: State machine of the morph that corresponds with the still images via colour-coding (yellow for states, green for transitions). Blue represents *signals* used to control the behaviour of transitions in the overall morph.

ABSTRACT

We present Deimos, a grammar for specifying dynamic embodied immersive visualisation morphs and transitions. A morph is a collection of animated transitions that are dynamically applied to immersive visualisations at runtime and is conceptually modelled as a state machine. It is comprised of state, transition, and signal specifications. States in a morph are used to generate animation keyframes, with transitions connecting two states together. A transition is controlled by signals, which are composable data streams that can be used to enable embodied interaction techniques. Morphs allow immersive representations of data to transform and change shape through user interaction, facilitating the embodied cognition process. We demonstrate the expressivity of Deimos in an example gallery and evaluate its usability in an expert user study of six immersive analytics researchers. Participants found the grammar to be powerful and expressive, and showed interest in drawing upon Deimos' concepts and ideas in their own research.

CCS CONCEPTS

• **Human-centered computing** → **Visualization theory, concepts and paradigms**; **Visualization toolkits**; **Mixed / augmented reality**.

KEYWORDS

Immersive Analytics, data visualisation, animated transitions, embodied interaction, user study, grammar

1 INTRODUCTION

Immersive environments, such as virtual and augmented reality (VR/AR), offer people a platform for human-computer interaction that utilises a variety of human senses and a range of physical human interactions. Compared to traditional desktop interaction, immersive environments offer users a more natural and *embodied* experience of interaction [14]. Affordances for interaction can be embedded directly within virtual objects, allowing people to use their bodies to physically act upon those objects in a manner that leverages proprioception [46]. In the same way real-world objects can morph and change shape in response to physical actions, so should embodied representations of data in Immersive Analytics [43]. Interaction is crucial in data visualisation to handle complexity and allow changes to views [47]. When an embodied visualisation is acted on by a user, it may undergo a transition in its visual state reflecting a change in encoding of data to representation. Animation is a very common technique to help users naturally keep track of such visual changes in statistical graphics [25, 50].

Animation that preserves congruency between changes in data and its visual representation [63] has been demonstrated to confer benefits in myriad situations. It can aid decision-making in certain tasks [22], increase viewer engagement in data-driven stories [2, 25], and promote literacy of unfamiliar and/or complex visualisation designs [53, 66]. However, these past explorations of animation in visualisation do not consider deep integration of animation and user interaction [72]. Since embodied interaction relies on gestural congruency between the interaction and resulting visual changes, interaction and animation both clearly go hand in hand for embodied Immersive Analytics applications.

However, compared to the decades of research and development of desktop-based data visualisation packages for animation (e.g. [19, 20, 32, 32, 62, 72]) and interaction (e.g. [4, 55, 72]), equivalent tools for Immersive Analytics lag far behind. While some Immersive Analytics research has investigated the combination of animation and interaction [38, 70], no work has yet presented a unified language and grammar for the definition of such immersive interactive animations. Moreover, despite the numerous toolkits supporting the authoring of immersive visualisations (e.g. [9, 11, 49, 58]), none allow for the rapid design and prototyping of embodied interactions: a glaring gap in the literature given the prevalence of embodiment in Immersive Analytics [6].

Therefore in this paper we introduce Deimos: a declarative grammar for authoring **d**ynamic **e**mbodied **i**mmersive **mo**rph**s** for immersive visualisations. We use the term morph to signify an embodied visualisation's ability to change shape when actions are performed on it by a user. In contrast to traditional animated transitions, morphs are *adaptive* and can be applied to any data visualisation in the environment that matches the partial visualisation specification of one of the morph's *states*. *Transitions* connect these states through animation that can be controlled by *signals*: data streams which stem from *embodied* user interaction. These are specified using the Deimos grammar and are written in JSON. The adaptivity of morphs allows them to be used in both analysis and presentation, depending on the degree of specificity of the morph.

We begin by detailing a set of design goals that allow morphs to leverage the strengths of immersive environments not present on desktops (Section 3). We then introduce the Deimos grammar itself, detailing its components, primitives, and specification (Section 4). Next, we describe a prototype implementation of the Deimos grammar (Section 5), developed in Unity as an extension to the DXR toolkit by Sicat et al. [58]. To demonstrate the expressivity of Deimos, we present an example gallery of morphs created in Deimos which highlights key characteristics of the grammar (Section 6). We also conducted a user study in which six Immersive Analytics researchers used Deimos to create their own morphs. Through semi-structured interviews with these participants, we gauge the usability of Deimos (Section 7) and elicit discussion topics and future research directions for morphs (Section 8).

Our contributions include both engineering efforts and theoretical knowledge, and are summarised as follows:

(1) A grammar for the declaration of dynamic, embodied, interactive animated morphs in immersive environments called Deimos, and an implementation of the grammar in Unity.
(2) An example gallery of interactive morphs, and a user study and semi-structured interview with six Immersive Analytics researchers that validates the design, implementation, and usability of the Deimos grammar.
(3) An open-source toolkit that enables rapid design and prototyping of embodied interactions for Immersive Analytics which can accelerate future research in this area.
(4) A conceptualisation of how morphs can be defined as keyframe animations but be later applied as presets & templates during analysis and/or presentation in VR/AR.
(5) A shift towards animation that is designed around and driven by (embodied) interaction, as opposed to existing methods that are mostly driven by the data.

2 RELATED WORK

2.1 Interactive Animated Transitions on 2D Screens

When a visualisation changes between visual states, animation is commonly used to help viewers maintain awareness of how data marks have changed throughout the transition [25, 50], thus minimising change blindness [47]. Various grammars and toolkits have been developed to aid designers in creating animated 2D statistical graphics for use in data-driven storytelling, such as Gemini [33] and Gemini^2 [32], Canis [20] and CAST [19], and DataAnimator [62]. These all fundamentally use keyframe animation, which has been shown to be the preferred paradigm of animation designers [61]. Earlier work by Tversky et al. [63] however could not find strong evidence of animated graphics being superior to static ones, especially as animations were often too complex or fast to be accurately perceived. They instead suggested that interactivity may be one way to capitalise on the strengths of animation by allowing users to directly control its playback (start, stop, rewind, etc.). Indeed, later research found that combining interactivity with animations can improve outcomes for certain data analysis tasks (e.g. [1, 51]). More recent work by Zong and Pollock et al. [72] formalised interactive

animation in the form of Animated Vega-Lite, an extension to Vega-Lite [55] which adds a time encoding channel and event streams to enable interactive animations for use in data analysis. Such interactive animations (e.g. [1, 51, 52, 72]) oftentimes expose their animation controls via a time slider and toggleable start/stop button. A good example of more direct interaction with conventional 2D animations is that of DimpVis by Kondo and Collins [34]. Through direct manipulation, users can touch a mark to select it, revealing a "hint path" that they can drag their finger along. This causes the visualisation to temporally navigate forwards or backwards using animation, with the selected mark following the hint path. The subsequent work on Glidgets by Kondo et al. [35] followed a similar premise but for dynamic graphs.

Of course, our work is differentiated from that of previous works by its immersive nature. We introduce new concepts and ideas to accommodate the shift to immersive environments, as we later detail in Section 3.

2.2 Embodied Interaction and Metaphors for Immersive Animations

Immersive Analytics is characterised by the use of interactive, engaging, and embodied analysis tools [43]. As such, there is a desire to move away from WIMP-based controls in favour of more direct, embodied styles of interaction [6, 12]. In embodied interaction [14], affordances are embedded within the artefact (in our case the data visualisation) itself, re-framing computational processes and operations as direct interactions of one's body with the physical world [16, 69]. This approach, as Dourish [14] notes, moves the user interface into the background where it is no longer the centre of attention. Embodied interaction is capable of leveraging metaphors [36], which can make it easier to remember interaction techniques and help users develop their mental model of the target domain [10]. Such metaphors have been extensively used in embodied Immersive Analytics research as a result, typically involving mid-air input. ImAxes by Cordeil et al. [12] used several interaction metaphors, such as direct manipulation to compose visualisations based on the proximity and relative orientation of embodied axes (a similar metaphor was also employed using the MADE-Axis by Smiley et al. [59]), and a "throw away" metaphor to delete these visualisations. FIESTA by Lee et al. [40] used a similar throwing metaphor but for pinning visualisations onto surfaces in the environment. FiberClay by Hurter et al. [27] used a "grab" metaphor for translating, rotating, and scaling a 3D trajectory visualisation.

Embodied interaction has also been used to directly control immersive animated transitions. Tilt Map by Yang et al. [70] is a visualisation that transforms between three states: a choropleth map, prism map, and barchart. As the visualisation is tilted using a VR controller, the visualisation is interpolated between the three states based on the tilt angle. More interesting is the recent work by Lee et al. [38] which demonstrated the use of the visualisation's spatial context as part of the metaphor. They described techniques for transforming visualisations between 2D and 3D, such as "extruding" a 2D visualisation into 3D using a "pinch and pull" gesture. For the technique to be valid however, the 2D visualisation must also be placed against a physical 2D surface. Through this, the metaphor is not only of the visualisation being extruded, but also of it being taken from a surface and "brought out into" space. Both of these works [38, 70] also demonstrate a high level of gestural congruency between the interaction and the visualisation that is manipulated, which is vital in embodied interaction [29, 30]. For example, the aforementioned extrusion technique described by Lee et al. [38] causes the visualisation to expand at the same rate as the hand is being pulled, directly mapping the extent of the extrusion to the user's hand position.

While other works do use animations in prototype implementations (e.g. [11, 18, 24]), animation has largely been used to maintain awareness during transitions and has not been the focal point of the research (unlike that of Yang et al. [70] and Lee et al. [38]). Therefore in this work we further explore the use of embodied interaction to control visualisation animations in immersive environments.

2.3 Toolkits and Grammars for Immersive Analytics

In recent years, many toolkits and frameworks have emerged to support research and development in Immersive Analytics. Some specialised toolkits have been developed which focus on specific application cases. MIRIA [7] allows user experiment data such as head and hand movements to be replayed in an AR environment for in-situ analytics. RagRug [17] is a situated analytics toolkit that updates immersive visualisations in either VR or AR through the use of a distributed data flow from the Internet of Things and NODE-Red.

Toolkits have also been developed to facilitate more generic visualisation authoring in immersive environments. While certainly not as mature as desktop-based packages such as gg2plot [67] and D3 [4], they typically provide a strong foundation that can and have been extended in subsequent works. These toolkits can largely be distinguished by how visualisations are created by the user. IATK [11] and u2vis [48] primarily expose their authoring tools through a GUI—typically through the Inspector window of the Unity game engine's editor. In contrast, DXR [58] and VRIA [9] facilitate visualisation authoring using human-readable JSON files. A grammar defines the syntactical rules of this JSON file, which is then interpreted by the system to produce the visualisation. In the case of both DXR and VRIA, the grammar is based on Vega-Lite's grammar [55]. Declarative grammars such as these have proven to be popular in data visualisation (e.g. [20, 33, 55, 72]) as they separate how a visualisation is defined from how it is created by the system. These declarative grammars can also make it easier to author data visualisations, thus leading to more rapid prototyping of ideas.

A common limitation in Immersive Analytics toolkits however is their support for interactivity. While toolkits like IATK [11] and DXR [58] provide built-in methods for interacting with the visualisation such as brushing and range filtering, they do not expose user-friendly means to create new interactions and instead require extending the source code itself. In contrast, our work aims to devise a grammar that can enable *interactive* animated transitions in immersive environments. As a result, our work contributes a grammar that can support both authoring of immersive animated transitions and help design new (embodied) interaction techniques.

3 DEIMOS DESIGN GOALS

The shift from 2D to 3D is more than just a third spatial encoding. Early in the development of Deimos, we identified several key differences between animated transitions in immersive and non-immersive environments that give rise to new research challenges. These challenges were rephrased and synthesised into three design goals (DG) which influenced the creation of the Deimos grammar, allowing us to focus on the novel characteristics of immersive headsets and environments, in turn opening up further design opportunities. Section 4 will explain the grammar itself and highlight how it addresses these design goals.

3.1 DG1: Morphs should be adaptable and flexible

Most animated transition grammars allow for rapid prototyping between the specification and the resulting animation. A low viscosity authoring process is particularly important when creating interactive animations for data analysis [72], allowing for fast and easy changes in the specification. The ability to rapidly prototype is facilitated by the constant access of keyboards for text input and pointing devices (i.e. mice) in desktop environments. In contrast, a challenge of immersive environments is that they often lack a convenient and comfortable form of text input that is required to write textual specifications, especially in VR or in highly mobile AR contexts. While a GUI can help facilitate this authoring process in VR/AR, designing a GUI is premature if there is no underlying grammar to support it, especially in such a novel environment.

To resolve this conflict, we take an approach inspired by Lee et al.'s recent work [38]. Many animated transition grammars treat transitions as a bespoke set of changes applied to visualisations predefined by the animation designer. Instead, we treat animated transitions as discrete operations that analysts can use to apply changes to their visualisations during their analysis. For example, the analyst might apply an animated transition that adds another spatial encoding to their visualisation, or converts a 3D barchart into a faceted 2D barchart. This turns animated transitions into a catalogue of adaptive and flexible operations that can be applied to immersive visualisations by analysts depending on the situation and goals. In this way, there exists two types of users of Deimos: immersive analytics system designers who use the grammar to create a catalogue of animated transitions in a desktop environment (e.g. Unity editor), and data analysts in VR/AR who use said animated transitions in their workflows and either do not have access to or are unfamiliar with the grammar. This necessitates a functional shift in grammar design, moving from highly tailored transitions with known data fields and encodings to generic transitions that operate on baseline idioms. As a result, any given transition specification can be reused across multiple visualisations, so long as they meet the baseline criteria specified by the author (e.g. be a barchart, have no z encoding).

3.2 DG2: Morphs should support embodied interaction

Animated transition grammars (e.g. [20, 33, 62]) have paid little attention to how transitions are triggered and controlled. In cases where these grammars do (e.g. [72]) it is limited to WIMP-style controls, with practitioners using similar input methods for their narrative visualisations (e.g. play button [52], linear slider/scroll [71]). In contrast, immersive environments rely on a completely different interaction paradigm which goes beyond the desktop and is both embodied (e.g. [11, 27]) and spatial in nature (e.g. [8, 26]). Novel language primitives are needed to support embodied interaction as existing ones (i.e. streams in Animated Vega-Lite [72]) do not adequately express relationships between entities, especially desktop-based grammars. One such relationship is that of the user and the visualisation itself: which part of the user is performing the interaction (e.g. hand, head), and which part of the visualisation contains the affordance to be interacted with (e.g. mark, axis). Spatial relationships and interaction also play a significant role in immersive environments [8, 26, 37]—which is not generally the case in non-immersive environments. For example, an immersive transition may be controlled based on the position of a handheld relative to a table [8]. By supporting this, immersive transitions become spatially aware. There can also be a relationship between the visualisation and its immediate environment, allowing immersive transitions to become context-aware [13, 60]. An example of this is the aforementioned "extrusion" techniques by Lee et al. [40] which require the 2D visualisation to be on a surface to be usable.

By expanding the Deimos grammar to support this paradigm, we enable a richer design space of visualisation transitions not otherwise possible on desktop environments, as they allow users to "reach through" and interact with their data in a more embodied and engaging manner [14]. It should be noted however that the actual design of such embodied interactions is left up to the end-users of Deimos. We decide not to enforce best practices in the grammar, such as the use of easy to understand metaphors [10, 36] and proper gestural congruency [29, 30]. Instead, we ensure Deimos is designed to allow said best practices to be followed—much in the same way that conventional programming languages do not enforce best practices.

3.3 DG3: Morphs should still support conventional approaches

While the two previous design goals are intentionally forward-thinking, we still want Deimos to be rooted in the same foundational elements as existing grammars. This is to both ensure that Deimos follows tried and true concepts and theories, and also to preserve a sense of familiarity for users of the grammar—especially for those new to immersive analytics. This includes the use of keyframe animation as the chief animation paradigm [62], the ability to specify timing and staging rules to customise the animation, and supporting WIMP-based interaction in hybrid immersive analytics setups or via immersive UX elements (e.g. [44]). Moreover, while DG1 advocates for generalised transitions that can be applied to a wide range of visualisations, Deimos should still allow for highly customised transitions that affect predefined visualisations created by designers. This is to allow animated transitions in Deimos to still be useful in controlled situations such as immersive data-driven storytelling. Therefore, our grammar should support both ends of two orthogonal spectrums: support both WIMP and embodied interaction to control and interact with animated transitions; and

support animated transitions that are either highly generalised and can apply to any visualisation, or highly specific and apply only to a particular visualisation in a controlled context.

4 THE DEIMOS GRAMMAR

Deimos is a declarative grammar used to specify *transitions* between *states* (keyframes), as well as the *signals* (interactions) used to control them. The grammar is largely based on the design goals listed in Section 3 and prior work by Lee et al. [38] on visualisation transformations. The Deimos grammar was developed in conjunction with its toolkit implementation (Section 5) through an iterative process. At each iteration, a working version of the grammar was defined and the toolkit was updated to support it. We created new example morphs at each iteration to test the new features added to the grammar, and maintained prior examples to validate any adjustments to the grammar (similar to unit testing). Many of these examples can be seen in Section 6. We continued this process until we felt that the grammar sufficiently met our design goals. The target audience of the grammar are developers and designers of immersive analytics systems. The morphs they create are then used by analysts in VR/AR.

A Deimos specification can formally be described as a three-tuple (elements suffixed with "?" are optional):

Morph := (states, signals?, transitions)

These components constitute what we call a *Morph*, the term signifying an embodied visualisation's ability to dynamically change shape and morph from one state to another via transitions upon matching certain conditions. A morph can be modelled as a state machine (Figure 2). A visualisation in the immersive environment only enters a morph's state machine when it matches one of its *states*. The state node that was matched with determines the possible *transition* nodes that it can access. These transition nodes are where changes are actually made to the visualisation, and are only entered when specified criteria are met. These criteria take the form of *signals*, which are streams of data typically generated by user interaction. They can also be used to control the behaviour of transitions themselves.

Morphs are an extension to any immersive visualisation authoring system already in place. That is, visualisations can still be manipulated in their usual way, but can have morphs applied to them should the relevant conditions be met. In this way, morphs serve purely to augment existing authoring techniques rather than supplanting them outright. When a visualisation is modified by the user in a manner external to the morph, it exits the morph state machine. It may then immediately re-enter following the same rules as before. A visualisation can have multiple morphs (and therefore state machines) active simultaneously. Multiple morphs can also be applied to the same visualisation concurrently, so long as the properties and encodings they affect do not overlap. The same morph specification can also be active across multiple eligible visualisations. This ability for the state machine to adapt to different visualisation configurations through a set of rules and conditions is what helps it satisfy DG1.

Morph specifications are written and stored as standalone JSON files. The use of JSON is very common amongst related grammars and allows for the separation between grammar and implementation (i.e. portability). A JSON schema provides auto-completion and tooltips for writing morph specifications with supported text editors. Figure 3 shows a basic example of a morph specification, and how it translates to the immersive environment and the state machine. The three main components of morphs are annotated with coloured boxes: states in yellow, signals in blue, and transitions in green. The same colour coding is used across all other figures. The rest of this section will explain in general terms what these components do.

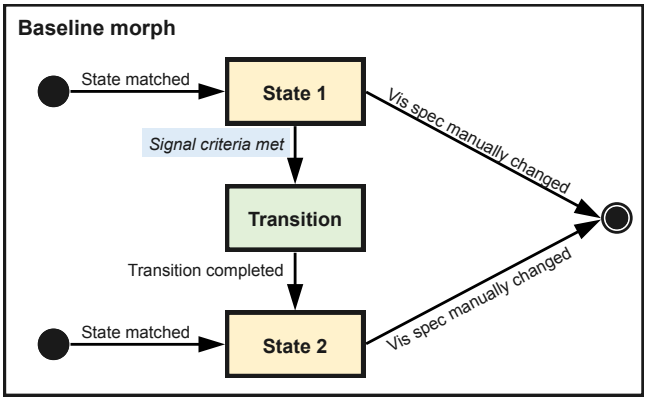

Figure 2: Baseline state machine for Deimos morphs showing a single unidirectional transition. More states and transitions can be added to the state machine with their own signal criteria, with support for bidirectional transitions.

4.1 States

A morph is comprised of at least two state specifications. A state can be defined by the following tuple:

state := (name, restrict?, partial visualisation specification)

The *name* property is a unique case-sensitive string used to reference this state specification in a transition Section 4.3). The *restrict* property is a Boolean that if set to `true` will remove the entry point associated with the state's node on the state machine Figure 3 for an example). This prevents a morph from starting at that state, making it only accessible via interconnecting transition(s). This is useful if it is illogical for a morph to start at that state, such as in unidirectional transitions. *Partial visualisation specification* is an arbitrary number of properties and components in the state object that all follow the same declarative notation as an actual visualisation. In other words, its syntax is the same as the visualisation package used in the system. For our implementation of Deimos, this is the DXR grammar [58] which in turn is based on the Vega-Lite grammar [55]. In the context of the DXR grammar, a partial specification can consist of any number of view-level properties (e.g. *mark*, *depth*) and/or encoding-level properties declared inside of an *encoding* component (e.g. *x*, *color*). The partial specification serves two purposes: (i) to determine if a visualisation matches (and therefore enters) this state; and (ii) to generate the keyframe used in the transition.

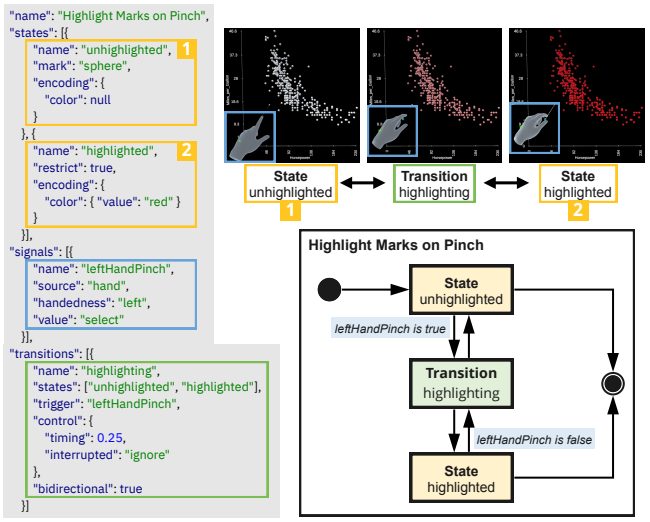

Figure 3: A basic example of a morph changes the mark colour of uncoloured visualisations to red whenever the left hand performs a pinch gesture. Colour-coded boxes denote the same component in different representations. Left: The morph specification. Top right: Still images of this morph being applied to a 2D scatterplot in an immersive environment. Bottom right: The state machine for this morph. The `"restrict": true` (shown in the left-hand box labelled with "2") prevents the morph from starting at the *highlighted* state, and `"bidirectional": true` (shown at the end of morph specification) allows the transition to function in both directions.

4.1.1 State matching process. Any visualisation properties specified as part of the *partial visualisation specification* in a state are used in the matching process against active visualisations. It is important to differentiate between the two types of specifications being used in this process: the visualisation specification created by the end-user, and the state specification (i.e. the *partial visualisation specification*) that exists as a part of the state component in a morph. Generally speaking, for a state specification to be matched against a visualisation specification, all properties defined in the former should also be defined in the latter, including their associated values. For example, if the state has `"color": {"type": "quantitative"}`, then the visualisation must also have a `color` encoding with the same `type` for it to match. As a rule of thumb, the fewer properties defined in the state specification, the more likely a visualisation can match successfully and have morphs applied to it. The opposite is also true, with more properties in the state specification making it less likely for any visualisation to match successfully. This effectively forms a spectrum. Morphs can be highly generic and can apply to many visualisations, allowing for adaptive morphs as per DG1. They can also only apply to specific datasets and field names, allowing for highly tailored morphs that are used in controlled environments as per DG3.

Deimos provides several primitives which affect the matching process that can be used in place of any JSON value in the state specification. They allow for more nuanced control over which visualisations can and cannot match, and are useful to prevent morphs from being accidentally applied to incompatible visualisations. Note that this is not an exhaustive set of primitives. While they were adequate for the purposes of this work, the grammar can easily be extended to include more if need be.

- **"*" (wildcard)**: The property should be in the visualisation but its value can be anything.
- **An inequality expression**: The property should be in the visualisation and its value should satisfy the inequality. Only applicable to numeric properties. e.g. `"value": ">= 100"`.
- `null`: The property should not be included in the visualisation regardless of its value.

4.1.2 Keyframe creation process. When a visualisation matches a state and one of its connecting transitions is activated, keyframes are generated for both initial and final states. These keyframes are used for actual animation during the transition. The initial keyframe is always the active visualisation's specification prior to the transition. No changes need to be made to it as it already matches the properties of the state itself. The final keyframe is created by modifying the initial keyframe using the following set of rules: (i) visualisation properties that are defined in the initial state but not in the final state are removed; (ii) properties that are not defined in the initial state but are defined in the final state are added; and (iii) properties defined in both states are set to the final state's value.

As with the state machine process (Section 4.1.1), Deimos provides primitives that can be used in place of any JSON value to refine the keyframe creation process. These primitives functionally act as placeholders which are later substituted with real values calculated at runtime, akin to the notion of variables. This allows for morphs to adapt to a wider range of situations without the need to hard-code field names, data types, etc. in morph specifications. For the purposes of the state matching process, all of these primitives are treated as wildcards. Their values are resolved after the keyframes have been created but before the transition is applied. Once again, this is not an exhaustive list of primitives and can easily be extended if need be.

- **JSON path accessor**: The value residing at the JSON path will be substituted into the property's value. Is either prefixed with "this." to access a property from this keyframe, or "other" to access a property from the other keyframe which is being transitioned to/from. e.g. `"x": "this.encoding.y", "field": "this.encoding.size.field"`.
- **A signal name**: The value emitted by the specified signal (Section 4.2) will be substituted into the property's value.
- **An expression**: The evaluated result of the expression will be substituted into the property's value. JSON path accessors and signal names can be used as variables. Only applicable to numeric properties. e.g. `"value": "other.encoding.size.value * 10"`.

All keyframes are stored throughout the entire lifespan of a morph. When the morph exits the state machine—the result of the associated visualisation having its specification manually changed

by the user (Figure 2)—all stored keyframes are deleted. Any added or changed properties will take their values from the state's keyframe if one already exists. The main purpose for this is to handle situations where a property is removed by a transition in one direction, but needs to be added back in by a transition in the reverse direction. Without stored keyframes, the removed property would no longer be known and therefore could not be added back in.

4.2 Signals

In Deimos, a signal is the resulting value from a stream of data captured from input events, drawing inspiration from Vega's signals [56] and event-driven functional reactive programming principles [65]. Signals can be used in Deimos to: (i) be substituted as values in keyframes (Section 4.1.2); (ii) act as conditional triggers that control when a transition actually begins (Section 4.3); and (iii) act as a tweening variable to control the progression of a transition (Section 4.3). No type safety is enforced in Deimos. A morph may contain zero or more signal specifications. Deimos has two main types of signals: signals that stem from some given source, and signals that evaluate a mathematical expression.

signal := sourceBasedSignal | expressionSignal

4.2.1 Source-based Signals. Source-based signals, as the name suggests, emit values from some input source. This is primarily from user interactions but could be extended to passively updating values from sensors, etc. We define two classes of source-based signals: deictic and non-deictic signals. Deictic signals express relationships between a source and target entity. While they mainly serve to model direct manipulation which is commonly associated with embodied interaction (DG2), they can also model situations where there is no actual direct contact. Non-deictic signals capture everything else, although these are mainly input sources that do not require some target/context to make sense (e.g. mid-air hand gestures, input source states, sensor data). Their production rules are:

sourceBasedSignal := nonDeicticSignal | deicticSignal
nonDeicticSignal := (name, source, handedness?, value)
deicticSignal := (name, source, handedness?, target, criteria?, value)

Both signal classes share the same three attributes. The *name* property references this signal in either a state (Section 4.1.2), an expression signal (Section 4.2.2), or a transition (Section 4.3). The *source* property denotes the type of source that values are to be retrieved from (e.g. `hand`, `head`, `vis`, `ui`). Certain sources can also specify the source's *handedness* to distinguish between `left`, `right`, or defaulting to `any`.

For non-deictic signals, the *value* property denotes what type of value to derive from the source, which is then emitted by the signal. This can either be the state of the user interaction (e.g. whether the hand is performing a `select` gesture) or the geometric properties of the source as an object in the immersive environment (e.g. `position` of the user's head). As previously mentioned, these are useful when some value of the input source is to be retrieved without it needing to be in the context of some other target or object. Figure 3 shows an example of a non-deictic signal: it does not matter what the hand is touching so long as it is performing the pinch gesture.

Deictic signals model relationships between entities, and are based on the interaction section of the design space by Lee et al. [38]. The *target* property denotes the type of object that the source is attempting to target. This can either be a part of the visualisation (e.g. `mark`, `axis`), a separate object in the environment (e.g. `surface`), or part of the user themselves (e.g. `head`). For the first two, a *criteria* property needs to be included to determine the logic used in selecting the target (e.g. `select`, `touch`, `nearest`). This logic is needed when there are multiple potential target objects that could be selected. Lastly, the *value* property can be used to derive three types of values. First, it can derive values from the *target* much in the same way as non-deictic signals do. For example, a `hand` source might target the `mark` that it is `selecting`, and the `position` of that mark is used as the value. Second, it can derive values from a comparison between the source and target. For example, a `vis` source might target the `surface` that it is `touching`, and the point of `intersection` between the vis and surface is used as the value. Third, a `boolean` value simply emits true if a target has been selected successfully, and false if no targets are selected.

Deictic signals in particular address the challenges in DG2 as they express relationships between entities, allowing morphs to react to direct interactions by the user (e.g. user's hand selects a mark). Of course, whether or not these interactions are truly embodied (i.e. it follows best practices) is dependent on how the morph designer uses deictic signals in conjunction with the grammar's other components. Deictic signals also allow morphs to be spatially-aware [8, 26, 37], as they can emit values that are based on spatial relationships between objects which can then be used to control the morph's behaviour (e.g. distance between user's head and the visualisation, orientation of two standalone tracked objects). Lastly, deictic signals allow morphs to become context-aware [13, 60], as they can emit values derived from a visualisation's relationship with its environment (e.g. is the visualisation touching a surface, is the visualisation close to a particular object). This may then act as conditionals to allow/disallow the morph from triggering (Section 4.3).

While not as critical to this work, the ability to facilitate WIMP-style interaction using these signals also helps fulfil DG3.

4.2.2 Expression Signals.

expressionSignal := (name, expression)

Expression signals allow for the arbitrary composition of signals using mathematical expressions. Their primary purpose is to modify and refine values emitted by source-based signals. We choose to use expressions as they allow arbitrary calculations to be performed in a familiar manner, instead of designing a completely new and potentially confusing domain-specific language. The *name* property references this signal in the same way as source-based signals. The *expression* property is a mathematical expression as a string. Basic mathematical operators can be used alongside select primitive functions (e.g. `normalise`, `distance`, `angle`). As with all other primitives, the list of supported functions can easily be extended. Any type of signal can be used as a variable by referencing its name. As previously mentioned, no type safety is enforced, meaning the user has to be aware of the data types present in the expression.

Expression signals are similar to deictic signals in that they help further address the challenges in DG2, but are more powerful in comparison. For example, while deictic signals only allow for a

single entity to be targeted, expression signals can combine two (or more) deictic signals together to calculate a new relationship between the targeted entities (e.g. distance between two marks selected by the user's hands).

4.3 Transitions

A morph is comprised of at least one transition specification. They functionally connect two state specifications together in the state machine (Figure 2). A transition can be defined by the following seven-tuple:

transition := (name, states, trigger?, control?, bidirectional?, disablegrab?, priority?)

The *name* property serves to identify this transition especially when multiple transitions are involved. The *states* property is an array of two strings, corresponding to the names of the initial and final states in the transition respectively. Referencing states via their name in this manner helps with encapsulation, keeping all state related syntax separated from the transitions. The *trigger* property is an equality expression that activates the transition when it evaluates as true, but only when the visualisation matches the initial state in the *states* property. The expression follows similar rules as expression signals (Section 4.2.2) but must return a Boolean value. Triggers are mainly used to let the user control when the transition is actually applied, usually as the result of some sort of input action or condition caused by the user. Not setting a trigger will cause the transition to be immediately applied when it enters the initial state. The *control* component is optionally used to further customise the behaviour of the transition. It is formally described by the following five-tuple:

control := (timing?, easing?, interrupted?, completed?, staging?)

The *timing* property controls the duration of the transition. If a number is used, the transition will interpolate between the two state keyframes over the given duration in seconds. Alternatively, the name of a signal can be used, in which case the signal will be used as the tweening variable *t*. This allows for the duration and direction of the interpolation to be controlled by the signal (and subsequently the user). In this situation, the transition will only begin when the signal is a value between 0 and 1, in addition to any other conditions. This defaults to 0 if not specified, which will result in jump cuts. The *easing* property applies an easing function to the transition, defaulting to a linear function if none is specified. Easing functions are commonly used in animations and help make animations look more natural. Functions that slow down the animation at the start and end can also make it easier to keep track of visual changes by making movement more predictable [15]. The *interrupted* property determines what happens when the *trigger* returns false whilst the transition is in progress. `initial` and `final` will cause the visualisation to immediately jump to the specified state. `ignore` will instead allow the transition to keep progressing until it naturally terminates. The `ignore` condition is particularly useful in cases where the *trigger* may inadvertently return false mid-transition but the transition should still continue, acting as a sort of fail-safe. This defaults to `final`. Similarly, the *completed* property determines what happens when the visualisation naturally terminates, either remaining at the `final` state or resetting back to the `initial` state instantaneously. Using the `initial` condition may be useful if the transition should not cause any long-term changes to the visualisation, particularly if the animation is alone sufficient to serve its purpose [38]. This also defaults to `final`.

The *staging* property allows for specific visualisation properties to be staged. Name-value pairs can be specified where the name is the property to be staged, and the value is an array of two numbers between 0 and 1 that correspond to start and end percentages. The property will only be animated when the transition period is within the given range. Any property not specified will not be staged. Staging is a common feature of animated transition grammars [25] and ours is no different. Note that the grammar does not support staggering.

The *bidirectional* property of the transition, if set to true (default false), allows the transition to start and end in the reverse direction. All transition settings remain the same, except the *trigger*, if specified, needs to return false in order for the reverse transition to activate. This serves mainly as a convenience function that prevents the need for two transition specifications to be written whenever a single bidirectional transition is desired. However, doing so is necessary in order to have distinct settings for either direction. The *disablegrab* property, if set to true (default false), will automatically disable the standard VR/AR grab action performed on the visualisation when the transition starts. This helps prevent visualisations from being inadvertently moved by the user when a transition's *trigger* uses a similar grab gesture. Lastly, the *priority* property can be used to handle edge cases where multiple transitions due to similar *trigger* conditions are activating on the same frame, but they conflict with the visualisation properties they modify. In this situation, the transition with the highest numbered priority will activate first, and all other conflicting transitions will be blocked. If priorities are equal, then the order in which they activate is random. The priority property defaults to 0.

4.4 Satisfaction of Design Goals

We now reiterate how our grammar satisfies the design goals listed in Section 3.

For DG1, the use of partial visualisation states (Section 4.1.1) and the keyframe creation process (Section 4.1.2) helps satisfy it. As the Deimos grammar is defined solely through JSON text, a library of generic morphs can be created in a development environment that has access to ergonomic text input (i.e. keyboards). When deployed in a production environment, the end-user in the immersive environment then has access to these (embodied) interactive morphs without needing to write the specifications themselves—a process which is notoriously difficult in VR and/or remote-AR environments. We provide a direct example of one such generic morph in Section 6.1. Establishing this JSON-based grammar also sets the foundation for designing a GUI that is intended for use in VR/AR, much in the same way that CAST [19] is the GUI implementation of the Canis grammar [20]. Through this, a morph author can rapidly prototype entirely in VR/AR.

For DG2, certain components such as deictic (Section 4.2.1) and expression signals (Section 4.2.2) directly support embodied interaction, as these signals listen to user input and/or changes in the entities in the environment and thus the relationships between them. As previously stated in Section 3.2, the grammar intentionally

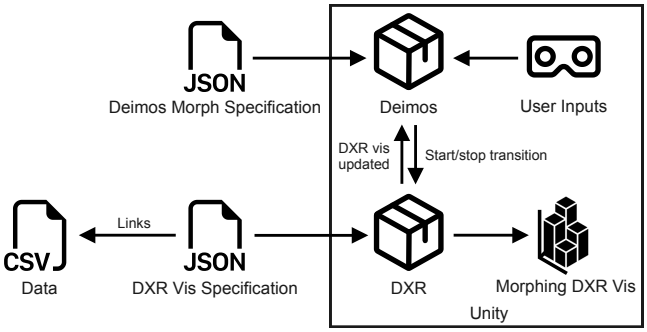

Figure 4: Overview of Deimos and how it interacts with our updated version of DXR [58]. A recreated version of the original DXR overview image is shown in the lower half.

does not enforce any best practices, including embodied interaction and animated transition principles. However, adherence to these guidelines is not isolated to any one component of a morph but instead across the entire specification. For example, even if direct manipulation is emulated through a deictic signal between the user's hands and the visualisation's marks, there would be little to no gestural congruency if the morph instead changed the visualisation's geometric size. Therefore, the ability of the grammar to express embodied interactions is dependent on the morph designer. We describe how a morph can use embodied interaction in a practical example in Section 6.2. We also describe how morphs can be rapidly iterated on in order to test new (embodied) interaction ideas in additional examples in Section 6.3.

For DG3, certain source signals (Section 4.2.1) allow for WIMP UI elements to be used to control morphs. This of course stands at odds with the embodied interactions of DG2, but our goal with Deimos is to support both ends of this theoretical spectrum. Section 6 as a whole contains multiple examples of these more conventional types of morphs.

5 DEIMOS IMPLEMENTATION AND TOOLKIT

We created a prototype implementation of the Deimos grammar using the Unity game engine in order to demonstrate its concepts and use. Deimos is open source, with its source code and documentation available on a public GitHub repository[1].

5.1 Data Visualisations

As Deimos is primarily an animated transition grammar, we need data visualisations to apply transitions to. We decided to use DXR by Sicat et al. [58] as the basis of our work. It is a toolkit developed for the Unity game engine designed for rapid prototyping of immersive data visualisations. The original DXR implementation provided support for an assortment of visualisation types, including scatterplots, barcharts, radial barcharts, and streamlines. These visualisations are specified in JSON files using an extended version of the Vega-Lite grammar [55], adding support for the *z* and *depth* encodings. We use DXR instead of other toolkits like IATK [11] as we found it easier to extend for our purposes. It already supports

[1]https://github.com/benjaminchlee/Deimos

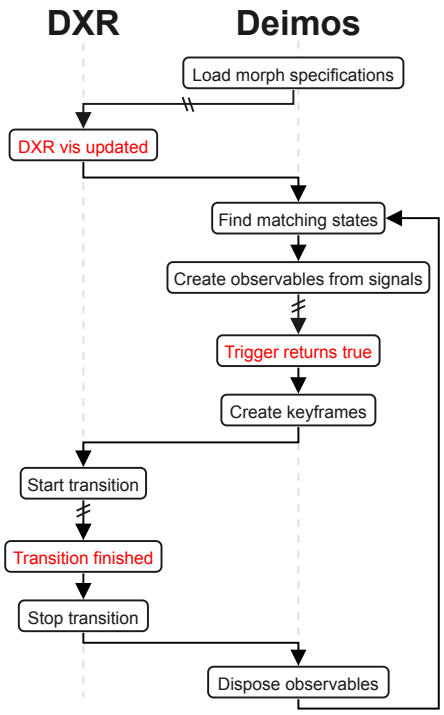

Figure 5: High-level overview of the Deimos pipeline and how it interacts with our updated version of DXR [58]. Red bubbles represent stages that wait for event(s) to fire before execution continues, also indicated by the preceding hatched arrow.

the Vega-Lite declarative grammar which is very popular in the visualisation community. DXR also uses individual GameObjects for each individual mark, simplifying mesh generation and management. This came at the cost of rendering performance however, especially when thousands of marks are displayed on the screen. To this end, we made performance improvements to how DXR instantiates and updates its marks and axes by introducing object pooling, especially since marks and axes may be modified multiple times in a morph. We also added several new visualisation types: choropleth and prism maps, stacked and side-by-side barcharts, and faceted charts (Section 6). However, as the original DXR implementation does not have support for data transformations like in Vega-Lite, neither does Deimos. This also means that animated transitions involving a time dimension (e.g. time varying scatterplots, barchart races) are not supported in Deimos.

5.2 Code Structure and Pipeline

Figure 4 provides an overview of Deimos' structure and how it interacts with our updated version of DXR. Morph specifications are contained in JSON files that are read by Deimos at initialisation. They can also be refreshed during runtime if the specifications are edited. Deimos interacts with DXR in two main ways. Deimos receives events from DXR whenever a visualisation has been updated, which includes the visualisation specification as an argument.

Deimos also sends start and stop function calls to DXR which executes the animated transitions.

Figure 5 provides a high-level overview of the Deimos pipeline in relation to DXR. While it is presented as a linear set of stages, the pipeline can reset or be exited in certain conditions. First, all morph specifications are read and loaded into Deimos. Next, whenever a DXR visualisation updates, Deimos is notified via event with the visualisation's specification. This specification is used to check against all state specifications in the loaded morphs using the rules in Section 4.1.1. For any state that has matched, observable streams are created for each signal that is part of the state's transitions, including trigger signals. Observables are created using the UniRx package [31] and are composed together where necessary. When a transition's trigger signal returns true (or if no trigger was specified in the first place), initial and final keyframes are created using the rules in Section 4.1.2. These two keyframes, along with other transition parameters such as tweening and staging variables, are sent to the relevant DXR visualisation to start the transition. When the transition has finished, Deimos stops the transition on the DXR visualisation. This step also updates the visualisation specification to reflect the new changes made by the transition. Deimos then disposes of all observables related to the transition. This process then starts anew again, with Deimos finding matching states to see if this newly updated visualisation is eligible for any morphs once more.

While Deimos is designed such that it exists separately from the visualisation framework used, they are still intrinsically linked to each other. Deimos is dependent on the visualisation framework to implement the actual animation and transition. It is also dependent on the grammar and syntax of the visualisations themselves. Therefore, translating Deimos to other visualisation toolkits requires adaptation to support the new declarative grammar, and the toolkit itself needs to support animation between keyframes via interpolation. While it is technically possible to create a middleware to translate visualisation specifications and thus increase modularity, we did not explore this option in this work.

5.3 XR Interactions

We use the Mixed Reality Toolkit (MRTK) [45] to enable XR interactions in Deimos. As a result, Deimos can be deployed on a range of platforms including Windows Mixed Reality, Oculus Quest, and HoloLens. However, due to the aforementioned performance limitations when working with large amounts of data, it is recommended to only use Deimos in tethered VR or remote rendering AR setups. Both controller and articulated hand tracking are supported in Deimos in the form of source-based signals (Section 4.2.1). While Deimos does not support eye gaze or voice input, these can be included in future work.

6 EXAMPLE GALLERY

We present several examples of morphs created with the Deimos grammar. We categorise and describe the examples in three ways, with the first two aligning with the design goals in Section 3. First, we highlight how morphs can be designed to adapt to different visualisation configurations using generic states (DG1), but also allow for bespoke morphs by using specific states in controlled contexts (DG3). Second, we demonstrate how morphs can be controlled using both embodied (DG2) and non-embodied (DG3) interaction methods. And third, we provide two scenarios in which Deimos can facilitate the prototyping of different interaction methods. All examples and their specifications are included in the Deimos Github repository. As such, we do not provide nor go into detail about each example's specification. The project files also contain additional example morphs not described in this paper.

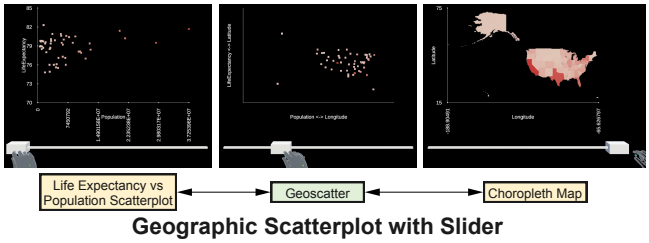

Figure 6: Still images of the *Geographic Scatterplot with Slider* morph, using Unity GameObjects as a slider to control the transition.

6.1 Generic vs specific morph examples

In DG1 and DG3, we described a spectrum in which morphs can vary between generic, adapting itself to a range of visualisation configurations, and specific, allowing it to be used in controlled settings.

On the generic end, we present the *3D Barchart Partitioning and Stacking* morph (shown in Figure 1). It takes a 3D barchart and either partitions it into a 2D faceted barchart, or stacks it into a 2D stacked barchart whenever it touches a surface in the immersive environment. During the transition, it also aligns the visualisation to be parallel against the surface that it had touched. This is an example of a morph involving three states and two transitions in a branch-like structure. The triggers are set up so that the applied transition is based on the angle of contact between the barchart and surface: orthogonal for the faceted barchart, and parallel for the stacked barchart. Its states are defined such that they only check that the encodings' types are correct (i.e. nominal x and/or z, quantitative y) and that it uses cube marks. Through this, so long as a visualisation is a 3D barchart then it can undergo this morph, greatly expanding the range of scenarios it can be used in. JSON path accessors are also used to substitute in the proper field names during runtime (i.e. *facetwrap*, *yoffset*).

On the other end of the spectrum, the *Geographic Scatterplot with Slider* morph (shown in Figure 6) demonstrates the use of two predefined states: a scatterplot and a choropleth map. Both of these are explicitly defined using exact encodings and field names (e.g. "Population", "LifeExpectancy"). Because of this, only a visualisation with these exact encodings and fields can undergo this morph. A transition connects the two states together, which is controlled using a linear slider represented by a Unity GameObject. A signal accesses the x position of this GameObject and uses it as the timing property of the transition. A morph like this is useful for controlled

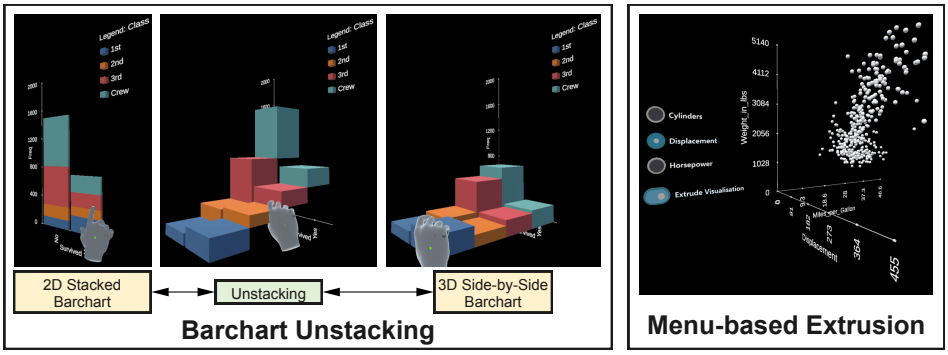

Figure 7: Examples of embodied and non-embodied morphs. Left: Still images of the *Barchart Unstacking* morph, using a "pinch and pull" gesture to unstack a 2D barchart into 3D. Right: The result of the *Menu-based Extrusion* morph showing the radial menu and toggle button.

settings like data-driven storytelling, as the visualisation(s) are all predefined by the author.

6.2 Embodied vs non-embodied morph examples

In DG1 and DG3, we described a spectrum in which morphs vary based on the use of embodied vs non-embodied (or WIMP-based) interactions.

On the embodied end, the *Barchart Unstacking* morph uses a "pinch and pull" metaphor as the gesture to unstack the bars of a 2D barchart into a side-by-side 3D barchart (shown in Figure 7 left). To strengthen the metaphor of bars being extruded out into 3D, a condition is added whereby the 2D barchart needs to be positioned against a surface for the morph to be allowed—introducing a contextual requirement to the morph. To initiate the transition, the user also needs to perform a pinch gesture on the visualisation itself, which is represented by a deictic signal. Other signals calculate the distance between the user's hand and the surface the visualisation is resting against. The transition uses this distance as its timing property, causing the bars to extrude at the same rate which the user pulls away from them. In this fashion, the user perceives themselves as actually stretching the barchart into 3D, thus resulting in a high level of gestural congruency [29, 30]. Of course, this is but one way in which embodied interaction can be achieved, but this approach can be replicated across other morphs to achieve similar styles of extrusion effects.

On the non-embodied end, the *Menu-based Extrusion* morph adds a third spatial dimension to a 2D scatterplot, but does so via an MRTK toggle button [45] (shown in Figure 7 right). A signal retrieves the state of this toggle button, and will trigger the visualisation when the button is toggled on. This example also demonstrates the use of a radial menu to select the field name of the newly added dimension. A signal retrieves the selected value and substitutes it into the 3D scatterplot state at keyframe creation. In comparison to the *Barchart Unstacking* morph, this example presents a much simpler and more familiar type of animated transition, albeit in an immersive environment.

6.3 Prototyping morph interactions

Lastly, we demonstrate how the grammar allows for signals to be easily swapped and modified to allow rapid prototyping of different interactions. In terms of the Cognitive Dimensions of Notations [23], this corresponds to a low level of *viscosity*.

In this example, we recreate *Tilt Map* by Yang et al. [70] using Deimos (shown in Figure 8 top). Three states are defined: choropleth map, prism map, and barchart. Two transitions are defined to connect these states linearly. A signal is then created to retrieve the tilt angle of the visualisation relative to the horizontal plane. This tilt angle is then subdivided into two ranges at specific angles using expression signals, that are then used as tweening variables for the two transitions (choropleth to prism, prism to barchart). With this, a visualisation will morph between the different states depending on its tilt. However, we can easily change the manner which the morph is controlled just by replacing the tilt angle with another source. A straightforward example is to replace it with the height of the visualisation relative to the floor (shown in Figure 8 bottom). The two expression signals which subdivide the range will also need to be updated to the new value ranges. In doing so we turn *Tilt Map* into a so-called "Height Map", just by changing a few lines in the morph specification. The result is shown in Figure 8.

Inspired by work on small multiple layouts in immersive environments [42], we created the *Proxemic-based Facet Curvature* morph (shown in Figure 9 top). It morphs into a faceted chart between three different layouts: flat, curved, and spherical. These three layouts correspond to three states in the morph, with two transitions connecting them linearly. A signal retrieves the distance between the user's head and the visualisation, with two more signals subdividing the distance into tweening variables (similar to the *Tilt Map* morph). As the user approaches the faceted chart, it begins to wrap around them into a curved layout, and when they are close enough it morphs into an egocentric spherical layout. This effectively makes the chart spatially aware of the user's position. To demonstrate another method of controlling this morph, we can replace the distance signal with the value of a rotary dial (shown in Figure 9 bottom). As the user rotates the dial the small multiples curve inwards or outwards. To do so, we create a separate cylinder GameObject in Unity which functions as this dial. We then replace

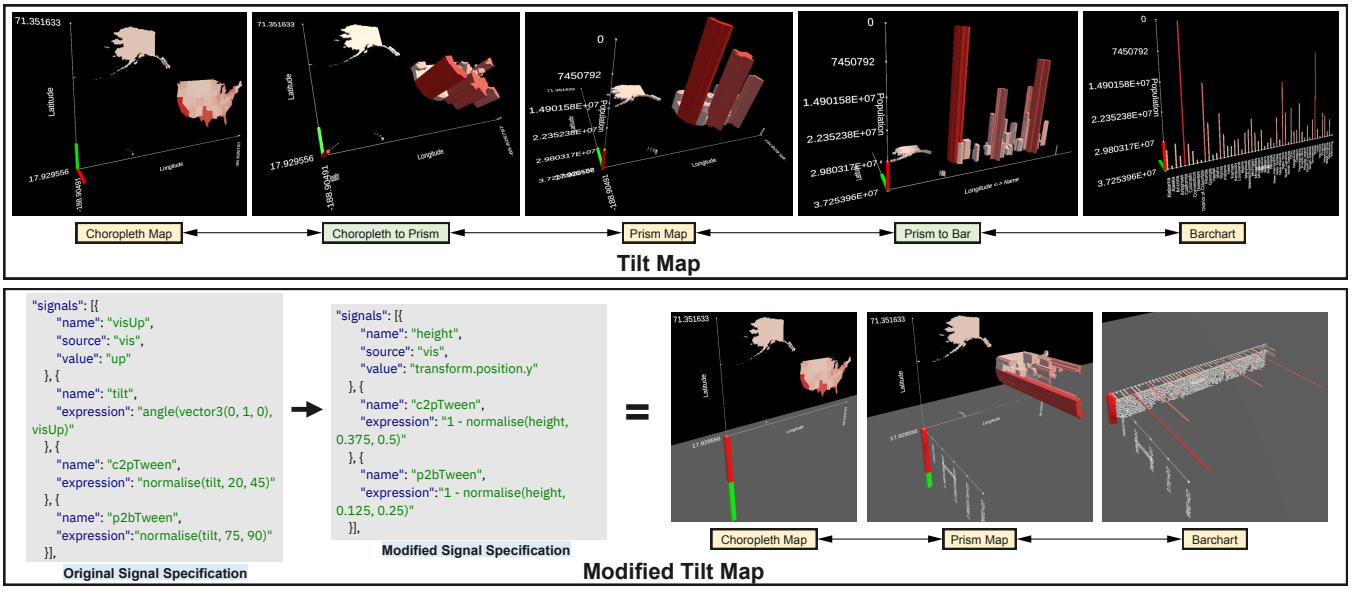

Figure 8: Top: Still images of the *Tilt Map* morph based on Yang et al. [70]. A red and green angle bracket is shown to provide rotation cues. Bottom: A modified version of *Tilt Map* showing changes to the signal specification and the resulting morph shown as still images. This example shows tilt being replaced with height. A red and green bar is shown to provide height cues.

the distance signal with a signal which retrieves the rotation value of the cylinder, and we also update the ranges of the two subdividing signals. This functionally turns the proxemics-based interaction into one involving the manipulation of an external object. This object is currently only virtual, but the concept can be applied to physical objects using either tangible input or motion tracking.

7 EXPERT EVALUATION

We evaluated Deimos in order to: (i) determine the ease of use and expressiveness of the grammar; (ii) get impressions on the concepts introduced in the grammar; and (iii) generate discussion topics and research directions on the use of animated transitions in immersive environments.

7.1 Study Design

We use an approach similar to Zong and Pollock et al. [72] by recruiting three developers of immersive analytics grammars and toolkits: Peter Butcher of VRIA [9], Philipp Fleck of RagRug [17], and Ronell Sicat of DXR [58]. To diversify our participant pool, we also recruited Zeinab Ghaemi of immersive geovisualisation [21], Tica Lin of embedded sports visualisation [41], and Jorge Wagner of the VirtualDesk exploration metaphor [64]. We hoped to learn how Deimos could be positioned within each researcher's respective works. To minimise learning requirements, we only invited researchers who have experience working with Unity.

The user study was conducted remotely in three sections, repeated for each participant. First, we conducted a 30-minute introductory session where we explained the goals of the study, demonstrated the examples in Section 6, and went through high-level concepts of the grammar. Second, we tasked participants to use Deimos unsupervised for at least 2.5 hours. They were given walkthroughs and documentation to learn the grammar, and were encouraged to create their own morphs with some suggestions given to them. This documentation can be found in the Deimos Github repository. Third, we held a one-hour semi-structured interview based on the aforementioned evaluation goals. We asked participants to show us their created morphs, whether they found the overall process easy or difficult, and what parts of the grammar they liked or disliked. For the three participants with toolkit development experience, we also asked how they would retroactively implement animated transitions in their respective toolkits, and if there would be any significant differences compared to Deimos and why. For the other three participants without toolkit development experience, we instead asked how Deimos could be used to support any part of their own research—if at all. However, we allowed the interview to diverge and continue organically, drilling down on any interesting comments participants may have made along the way. Throughout the study period, we modified the documentation based on participant feedback. While we made bug fixes to Deimos where necessary, we did not add or change any features. Each participant was offered a AU$150 gift card as compensation for their time.

The interviews were recorded and transcribed. The first author independently performed thematic analysis [5] on all six transcriptions, with two other authors doing the same on three transcriptions each. These three authors then discussed and synthesised the main themes together, which form the structure of this section and the following Discussion section.

7.2 Usability feedback

We compile participant feedback based on a selection of the most relevant Cognitive Dimensions of Notations [23]. Rather than using the dimensions as heuristics—a common approach in related works

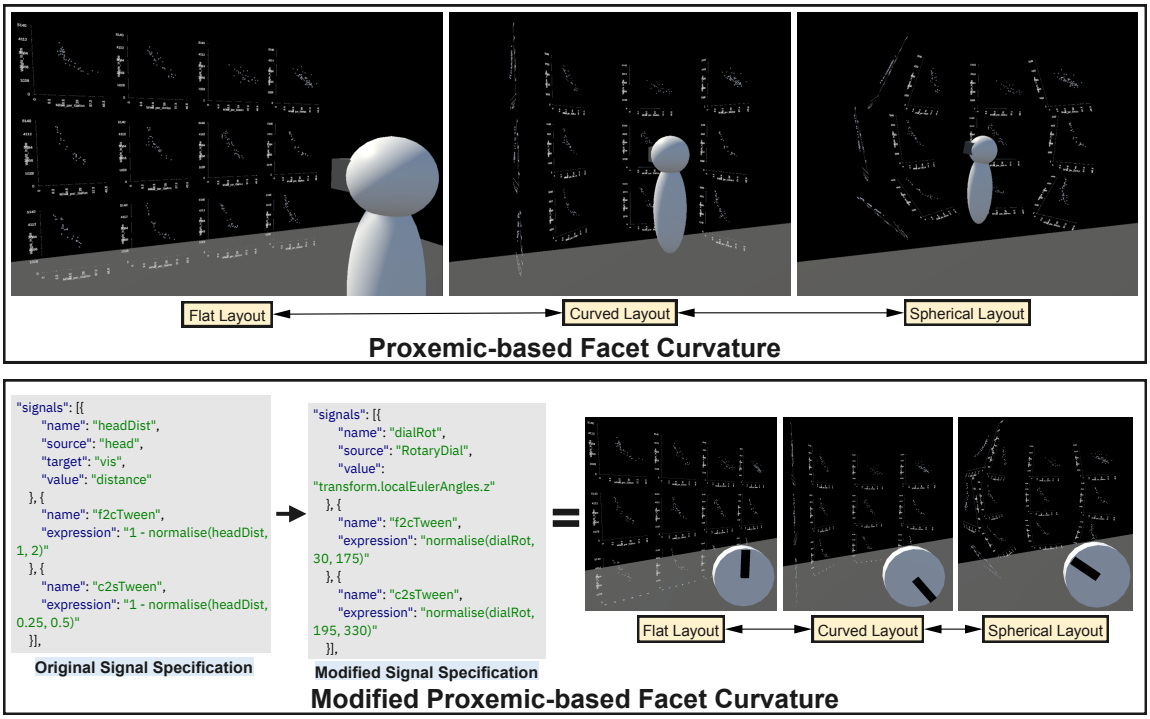

Figure 9: Top: Still images of the *Proxemic-based Facet Curvature* morph, which curves around the user based on the distance between them and the chart. Bottom: A modified version which replaces distance with the rotation of a separate dial object. The changes to the signal specification are shown with the resulting morph shown as still images.

(e.g. [54, 56])—we use them from a usability perspective to evaluate the Deimos grammar. However, we provide self-evaluation for certain dimensions where relevant.

Error proneness (likelihood of making errors). All participants spent the required 2.5 hours using the toolkit, however four of the six spent 7–8 hours using it. The initial reasoning given by most participants was that they enjoyed their time with Deimos and learning how it worked. On further inspection however it was clear that this was in part due to the steep learning curve of the grammar, with Fleck commenting "I don't feel that three hours are enough." We identified several potential causes of this, largely due to grammar's *error proneness*. First, many participants (Fleck, Ghaemi, Lin, and Wagner) were unfamiliar with the DXR grammar, with even Sicat not having used DXR for three years. As a result, two grammars needed to be learnt, naturally increasing learning time. As the Deimos grammar is intrinsically linked to its visualisation grammar (Section 5.1), it is apparent that the choice of visualisation package brings not only technical but also notational difficulties. Second, our documentation assumed full knowledge of Unity and its functions which not all participants had. Third, the error messages provided by the Deimos prototype were not useful for participants. While the JSON schema validates whether the morph specification is syntactically correct before it is parsed, no check exists for semantic correctness (e.g. making sure *name* properties are unique). This has since been corrected in the prototype. Some participants suggested ways of easing the learning curve. Sicat suggested video tutorials to better explain the grammar, whereas Butcher suggested providing the DXR documentation as pre-reading before the study is even conducted. Interestingly, no participant suggested changes to the grammar itself beyond simple name changes (the terms *signals* and *restrict*). Whether this is due to participants not having had enough time to be exposed to Deimos' advanced features is unclear.

Closeness of mapping (closeness to problem domain). The lack of grammar changes suggested by participants could be at least partially explained by its *closeness of mapping*. All participants, when asked, had little to no issues understanding how the grammar models the state machine (Figure 2). The only participant who raised potential challenges was Fleck, citing the differences between declarative and imperative languages. As Unity primarily uses imperative programming, the shift to a declarative style in Deimos could confuse certain users, particularly when constructing an interaction using signals. We do not believe this to be a major issue however, especially if the immersive visualisations also use a declarative language (e.g. DXR [58], VRIA [9]).

Viscosity (resistance to change). After following the walkthroughs, all participants used the same strategy of combining parts of existing examples together to create new morphs to facilitate their learning. For example, Wagner combined the states and transitions of *Tilt Map* example and the signals of the *Proxemic-based Small Multiple Curvature* example to create a rudimentary "Proxemic Map". There are only a few examples of participants extending existing examples with completely new components: Sicat remapped the proxemic interaction of the *Proxemic-based Small*

Multiple Curvature example with a virtual rotary dial (the same as in Section 6.3), and Butcher created a stacked barchart to side-by-side barchart morph based on whenever the mouse is clicked. These all demonstrate a low level of *viscosity* within the grammar, as participants were generally able to achieve their goals without issue (minus the aforementioned issues regarding error proneness). The same concept was also described in Section 6.3. However, poor error messages introduced viscosity for a few participants. For instance, Lin had tried to create a reduced version of the *3D Barchart Partitioning* example by removing all surface related signals, but the toolkit did not warn her to remove the references to these signals in the states, resulting in errors. This need to keep track of changes in multiple parts of the specification contributes to higher viscosity.

Visibility (ability to view components easily). Several participants (Fleck, Sicat, and Ghaemi) noted issues relating to the *visibility* of signals in the grammar, primarily due to the large number of possible keywords involved. It was not obvious what options and/or combinations of signals are available without resorting to the documentation, although the JSON schema aided this process. The same participants acknowledged however that this reliance on documentation is fairly normal for toolkits, especially with only a few hours of experience. From a technical perspective, the Deimos prototype improves visibility by exposing the names of any active morphs and/or transitions on each visualisation, and provides a toggle to print the emitted values of signals to the console for debugging purposes. Further debug messages can also be enabled which show the visualisation specifications of generated keyframes in JSON format. While these features were not explained in the documentation, they were highly useful during the development of Deimos and the creation of our example gallery.

8 DISCUSSION

This section continues from Section 7 by summarising the main themes and discussion topics of the semi-structured interviews with our expert participants. We also include several adjacent topics to round out the discussion of immersive morphs—especially in the context of other animated transition grammars.

Adaptive morphs. While some participants liked the concept of adaptive morphs, others found it getting in the way of their authoring process. Butcher saw value in adaptive morphs, saying "I could see why that would be useful, especially if you had a large array of different charts... having it modular just makes sense." Wagner thought that "the premise works well", but clarified that he would prefer to have "a [morph] specification for each type of graph" instead of one hyper-generic morph that applies to all visualisation idioms. Ghaemi was caught off-guard by this function when her new morph was being applied to other visualisations unintentionally (a result of overly generic states), but was able to reason with modifying the states to ensure that they are more specific. Fleck and Sicat faced a similar issue, but instead suggested the ability to use an ID to directly target a specific visualisation, skipping the state matching process altogether. This was particularly of relevance to Fleck, where in *RagRug* [17] "the user does not create a visualisation [themselves], but the system creates the existing visualisations." Overall, participants were able to grasp the concept of adaptive morphs, but it is apparent that their experiences come from the perspective of the morph author. A quantitative evaluation involving data analysis utilising pre-made morphs for practical tasks would be needed to fully evaluate the concept.

The purpose of morphs. All participants found the examples exciting and interesting, but some had thoughts on their actual purpose. Ghaemi said that morphs are mainly useful when they add or change the data shown, rather than simply remapping encodings (e.g. *Stacked Barchart Extrusion* example). Lin similarly said that she would only use morphs when working with large amounts of data, such as combining proxemics with Shneiderman's mantra [57], or when working with multiple views, but "if it's only one smaller data set, and one chart, I probably wouldn't use it to morph between different columns." Butcher said that while our example morphs were "neat and novel", their animations did not strictly reveal new information, such as a time-varying scatterplot does. Therefore, future work should investigate specific use cases for morphs and how morphs may potentially vary between them.

Embodied interaction and discoverability. The reception to the use of embodied interactions in Deimos (DG2) was positive, but two participants raised discussion topics around their long-term effects. Many of our example morphs use interaction metaphors for embodied interaction (e.g. collide with surface, pinch and pull). Sicat expressed concern over the use of these metaphors, saying "...maybe in my application, pinning to the wall means or does something, and then someone else develops a morph where stick to the wall does something else... that might confuse people... there's no universal rule that says, pinning to the wall should do this." When asked if Deimos could play a role in shaping these metaphors, Sicat responded "I would keep it open for now and just let [researchers] explore", noting that the field is still not mature yet. He then suggested the use of tooltips to guide users in discovering morphs, especially when conflicting metaphors are used, but stated this is of low priority. In a similar vein, Lin suggested two ways of improving embodied morphs and their discoverability, especially as she had difficulties performing the rotation required for the *3D Barchart Partitioning and Stacking* example. The first was to have the system predict what action the user is about to do, and display the morphs associated with that action in a "gestural menu" that the user can select to trigger the morph. The second was to show a preview of the morph while performing the interaction. When asked about the importance of these features, she said that they "probably [do not] affect the current grammar, because it's more like an assistant towards the completion of certain interactions", and that they are more like external scripts loaded after the core grammar. Overall, while there are broader implications of the use of embodied interaction in immersive analytics, we see the power in Deimos being used to explore this design space in the long term, rather than immediately prescribing them in this work.

GUIs and morph templates. Fleck, Sicat, and Ghaemi brought up ideas on how GUIs can be incorporated into Deimos. Fleck suggested the use of data flows in Node-RED to author morph specifications in JSON, similar to how visualisation specifications are created in *RagRug* [17]. Sicat recalled his own experiences developing DXR's GUI [58], noting that a GUI can be useful for non-experts and even end-users to create their own morphs. In a similar vein, Ghaemi said that a GUI would have greatly assisted her learning process with Deimos, citing her lack of experience in both DXR

and toolkits in general. However, both participants clarified that the GUI should only cover basic functions, and advanced features should only be accessed in JSON format. Sicat went on to suggest that the GUI could expose templates for different parts of the grammar that allows users to mix and match and create new morphs, which would be exposed through dropdowns and menus. He compared this idea to how he used the grammar himself, saying "I went through your examples, copied the morphs and then pasted it into my morphs and then just modified them a bit. So it's kind of [the] same idea, right? Just a different interface. So for non-experts [it] would be super easy." Lin suggested something similar except from an interaction perspective, especially as in our included examples "the interaction you perform is very standardised." In other words, a set of template interaction techniques could be provided to accelerate the morph authoring process. This feedback opens many future design possibilities for how a GUI for toolkits like Deimos might look like, especially if it can allow end-users in VR or AR to create and/or modify their own morphs to suit their own needs without needing to write JSON.

Inspiration drawn from the toolkit. All participants drew interesting comparisons between Deimos and their respective works. Wagner, Ghaemi, and Lin all showed great interest in morphs that transition between 2D and 3D. For Wagner, from the context of his work on VirtualDesk [64], said "it would be very interesting to be able to just snap [3D visualisations] to the desk, and then they project to 2D, which is something that many experts are very comfortable with, but then I could show to them that they can extract [the visualisation] from the desk or from the wall, and try to grab it and look around..." For Ghaemi whose field is immersive geovisualisation [21], it was to have the morph directly tied to adding layers to a virtual map, "[when the] 3D chart collides with the map, the bars could be scattered through the buildings, so I can see the charts on top of the building." For Lin, she raised ideas in the context of embedded sports visualisation [41], whereby "you [can] drag the 2D charts onto a specific player, or maybe drag it onto the court, like the flat ground floor, and then it just suddenly morphs into this heatmap." In this sense, rather than a visualisation just morphing between 2D and 3D, it could also morph between being embedded and non-embedded [68]. We then asked whether they could see themselves using Deimos to aid in their research. Wagner thought that as a proof of concept it would work "super well", but cited the poor scalability of the toolkit as a reason against using it. Ghaemi was receptive, hypothesising that "the [toolkit] that you have it's, at least, for some of [my ideas], I'm pretty sure that I can implement what I want." She also noted that there are no other immersive analytics toolkits that currently enable animated transitions in the manner she desired. Lin said "there's a high chance that I could use this library to help me prototype some scene to show [sports analysts and coaches]." After this proof of concept stage however, she would instead develop her own research prototype from the ground up to support specific features such as "instant data updating". Lastly, Butcher said that "seeing the change in data and understanding what you know, getting something out of it, it's important... certainly not enough attention has been paid to it in the past I don't think, especially in the immersive space." He followed this up by saying "it's definitely something we're going to look at in future for sure, the effect is fantastic." While it is expected that not every researcher can make use of the Deimos grammar and the toolkit, our user study clearly demonstrates the significance of this work in generating further research ideas and promoting the study of animated transitions in immersive analytics.

Animation authoring paradigms. Deimos was originally designed around keyframe animation as its main authoring paradigm. Interestingly, Deimos can technically be seen as having a combination of both keyframe and preset & templates paradigms. This is arguably a good thing, as Thompson et al. [61] recommend authoring tools to combine multiple paradigms together to accommodate differences in designers' preferences. In truth, our use of the two paradigms is actually dependent on who is using the morph. In Section 3.1 we described two types of users of Deimos: the person who is creating the morph in a development environment (i.e. the "morph author"), and the person who is actually using the morph in an immersive environment (i.e. the "end-user"). The morph author creates the morph with a keyframe mindset, and the end-user uses the morphs as though they were presets & templates. Of course, when used for data exploration the VR/AR analyst does not necessarily need to interpret morphs as presets. Much like Data Clips [3] allows for data videos to be created using preset clips however, it is theoretically possible to re-frame Deimos in a similar manner. Morph authors create preset morphs that apply to generic states. End-users then combine these preset morphs together to create linear narratives or non-linear experiences. While this is merely speculative, we believe that future research can consider and further investigate this unique combination of authoring paradigms for animated transitions.

Data-driven vs interaction-driven animation. Deimos stands apart from other works in the manner in which animations are initiated and viewed by end-users once they are defined. Animations in Animated Vega-Lite [72], Canis [20], Data Animator [62] and so on are more data-driven. Specifications are tailored around the intricacies of the loaded dataset, with grammars like Gemini [33] and Gemini^2 [32] even providing recommendation systems to further improve the animations created. Completed animations are then passively viewed by the end-user, with little to no input required to initiate and/or control its playback. In contrast, Deimos is a more *interaction*-driven grammar. Morph specifications consider not only the change in visual encodings, but also how the user interacts with the system to trigger the morph itself. Completed morphs are then *actively* viewed by the end-user, with them potentially having a high degree of control over the morph's playback and function. This difference is intentional, as immersive environments are inherently more interactive and embodied [43] than desktop environments, encouraging users to "reach out" and directly manipulate their data. We expect and encourage future research on animations in Immersive Analytics to maintain this interaction-driven mindset—even for presentation and storytelling to better engage and immerse users through interactivity [28, 39].

9 LIMITATIONS

Our work naturally has several limitations in regards to the grammar, the technical implementation, and the user study. First, our grammar is built upon several key concepts such as dynamic morphs and embodied interaction. While we aimed to justify these ideas in

Section 3, we did not properly evaluate them with actual end-users in VR/AR performing data analysis tasks. Therefore, we cannot confidently say that our approach is quantifiably beneficial for immersive analytics. Second, our participants were not exposed to all of the functionalities of Deimos. It is certainly possible that there are pain points when using Deimos' advanced functionalities which were not identified due to the limited amount of time participants spent using it. This could include the inability to perform certain embodied gestures with the grammar, or difficulties managing morphs that contain more than 2 or 3 states and/or transitions. Third, as the grammar is dependent on the visualisation package that it is built upon, many of its limitations are born from DXR [58]. Limitations include the inability to transition between different mark types, lack of runtime data transformations, and overall poor scalability compared to other toolkits like IATK [11] especially when rendering large amounts of data. The inability to transform data (e.g. aggregation and filtering) is especially troublesome as it meant that time-varying animations (e.g. Gapminder [52]) were not considered while designing the grammar, and using certain visualisations in morphs such as barcharts required pre-processing. While we had attempted to add data transformations into DXR ourselves, the challenges in using .NET as a scripting language made it difficult to achieve a syntax remotely equivalent to that of Vega-Lite [55]. We see this as obvious future work, especially as it can allow visualisations to not only morph between encodings, but also between different levels of aggregation, filters, or even different datasets.

10 CONCLUSION

This paper presented Deimos, a grammar and toolkit for prototyping morphs in immersive environments. Morphs are a collection of animated transitions that occur between different defined states, which are triggered and modified by the use of signals. These morphs are dynamically applied to visualisations during runtime, and are capable of leveraging embodied interaction to enable interactive animated transitions. We view Deimos as an initial foray into what a grammar to create embodied animated transitions in immersive environments would look like. While our example gallery and user study demonstrated Deimos' ability to create a wide range of morphs, future work would seek to understand how these morphs are used by actual data analysts and/or audiences of immersive data stories in VR/AR. We also hope that this work fuels greater interest in the use of dynamically morphing embodied visualisations in Immersive Analytics.

ACKNOWLEDGMENTS

We thank the six Immersive Analytics researchers who took part in our user study: Peter Butcher, Philipp Fleck, Zeinab Ghaemi, Tica Lin, Ronell Sicat, and Jorge Wagner. We also wish to thank our anonymous reviewers for their valuable feedback, and Jiazhou Liu for his assistance during a pilot study.

REFERENCES

[1] Felwa A. Abukhodair, Bernhard E. Riecke, Halil I. Erhan, and Chris D. Shaw. 2013. Does Interactive Animation Control Improve Exploratory Data Analysis of Animated Trend Visualization?. In *IS&T/SPIE Electronic Imaging*, Pak Chung Wong, David L. Kao, Ming C. Hao, Chaomei Chen, and Christopher G. Healey (Eds.). Burlingame, California, USA, 86540I. https://doi.org/10.1117/12.2001874

[2] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Jason Leboe-McGowan, and Pourang Irani. 2018. Hooked on Data Videos: Assessing the Effect of Animation and Pictographs on Viewer Engagement. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*. ACM, Castiglione della Pescaia Grosseto Italy, 1–9. https://doi.org/10.1145/3206505.3206552

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

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

[5] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[6] Wolfgang Büschel, Jian Chen, Raimund Dachselt, Steven Drucker, Tim Dwyer, Carsten Görg, Tobias Isenberg, Andreas Kerren, Chris North, and Wolfgang Stuerzlinger. 2018. Interaction for Immersive Analytics. In *Immersive Analytics*, Kim Marriott, Falk Schreiber, Tim Dwyer, Karsten Klein, Nathalie Henry Riche, Takayuki Itoh, Wolfgang Stuerzlinger, and Bruce H. Thomas (Eds.). Springer International Publishing, Cham, 95–138. https://doi.org/10.1007/978-3-030-01388-2_4

[7] Wolfgang Büschel, Anke Lehmann, and Raimund Dachselt. 2021. MIRIA: A Mixed Reality Toolkit for the In-Situ Visualization and Analysis of Spatio-Temporal Interaction Data. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–15. https://doi.org/10.1145/3411764.3445651

[8] Wolfgang Büschel, Patrick Reipschläger, Ricardo Langner, and Raimund Dachselt. 2017. Investigating the Use of Spatial Interaction for 3D Data Visualization on Mobile Devices. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. ACM, Brighton United Kingdom, 62–71. https://doi.org/10.1145/3132272.3134125

[9] Peter W. S. Butcher, Nigel W. John, and Panagiotis D. Ritsos. 2021. VRIA: A Web-Based Framework for Creating Immersive Analytics Experiences. *IEEE Transactions on Visualization and Computer Graphics* 27, 7 (July 2021), 3213–3225. https://doi.org/10.1109/TVCG.2020.2965109

[10] John M. Carroll, Robert L. Mack, and Wendy A. Kellogg. 1988. Chapter 3 - Interface Metaphors and User Interface Design. In *Handbook of Human-Computer Interaction*, MARTIN Helander (Ed.). North-Holland, Amsterdam, 67–85. https://doi.org/10.1016/B978-0-444-70536-5.50008-7

[11] Maxime Cordeil, Andrew Cunningham, Benjamin Bach, Christophe Hurter, Bruce H. Thomas, Kim Marriott, and Tim Dwyer. 2019. IATK: An Immersive Analytics Toolkit. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, Osaka, Japan, 200–209. https://doi.org/10.1109/VR.2019.8797978

[12] Maxime Cordeil, Andrew Cunningham, Tim Dwyer, Bruce H. Thomas, and Kim Marriott. 2017. ImAxes: Immersive Axes as Embodied Affordances for Interactive Multivariate Data Visualisation. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, Québec City QC Canada, 71–83. https://doi.org/10.1145/3126594.3126613

[13] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction* 16, 2 (Dec. 2001), 97–166. https://doi.org/10.1207/S15327051HCI16234_02

[14] Paul Dourish. 2001. *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press, Cambridge, Mass. http://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=6267252

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

[16] Kenneth P. Fishkin, Anuj Gujar, Beverly L. Harrison, Thomas P. Moran, and Roy Want. 2000. Embodied User Interfaces for Really Direct Manipulation. *Commun. ACM* 43, 9 (Sept. 2000), 74–80. https://doi.org/10.1145/348941.348998

[17] Philipp Fleck, Aimee Sousa Calepso, Sebastian Hubenschmid, Michael Sedlmair, and Dieter Schmalstieg. 2022. RagRug: A Toolkit for Situated Analytics. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–1. https://doi.org/10.1109/TVCG.2022.3157058

[18] Flow Immersive, Inc. 2022. Data Storytelling with Immersive Visualizations. https://flowimmersive.com

[19] Tong Ge, Bongshin Lee, and Yunhai Wang. 2021. CAST: Authoring Data-Driven Chart Animations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–15. https://doi.org/10.1145/3411764.3445452

[20] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang. 2020. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum* 39, 3 (June 2020), 607–617. https://doi.org/10.1111/cgf.14005

[21] Zeinab Ghaemi, Ulrich Engelke, Barrett Ens, and Bernhard Jenny. 2022. Proxemic Maps for Immersive Visualization. *Cartography and Geographic Information Science* 49, 3 (May 2022), 205–219. https://doi.org/10.1080/15230406.2021.2013946

[22] Cleotilde Gonzalez. 1996. Does Animation in User Interfaces Improve Decision Making?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Common Ground - CHI '96*. ACM Press, Vancouver, British Columbia, Canada, 27–34. https://doi.org/10.1145/238386.238396

[23] T. R. G. Green. 1989. Cognitive Dimensions of Notations. In *People and Computers V*. University Press, 443–460.

[24] Devamardeep Hayatpur, Haijun Xia, and Daniel Wigdor. 2020. DataHop: Spatial Data Exploration in Virtual Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 818–828. https://doi.org/10.1145/3379337.3415878

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

[26] Sebastian Hubenschmid, Johannes Zagermann, Simon Butscher, and Harald Reiterer. 2021. STREAM: Exploring the Combination of Spatially-Aware Tablets with Augmented Reality Head-Mounted Displays for Immersive Analytics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–14. https://doi.org/10.1145/3411764.3445298

[27] Christophe Hurter, Nathalie Henry Riche, Steven M. Drucker, Maxime Cordeil, Richard Alligier, and Romain Vuillemot. 2019. FiberClay: Sculpting Three Dimensional Trajectories to Reveal Structural Insights. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 704–714. https://doi.org/10.1109/TVCG.2018.2865191

[28] Petra Isenberg, Bongshin Lee, Huamin Qu, and Maxime Cordeil. 2018. Immersive Visual Data Stories. In *Immersive Analytics*. Springer International Publishing, Cham, 165–184. https://doi.org/10.1007/978-3-030-01388-2_6

[29] Mina C. Johnson-Glenberg. 2018. Immersive VR and Education: Embodied Design Principles That Include Gesture and Hand Controls. *Frontiers in Robotics and AI* 5 (2018). https://www.frontiersin.org/articles/10.3389/frobt.2018.00081

[30] Mina C. Johnson-Glenberg and Colleen Megowan-Romanowicz. 2017. Embodied Science and Mixed Reality: How Gesture and Motion Capture Affect Physics Education. *Cognitive Research: Principles and Implications* 2, 1 (May 2017), 24. https://doi.org/10.1186/s41235-017-0060-9

[31] Yoshifumi Kawai. 2022. UniRx - Reactive Extensions for Unity. https://github.com/neuecc/UniRx

[32] Younghoon Kim and Jeffrey Heer. 2021. Gemini ^{2} : Generating Keyframe-Oriented Animated Transitions Between Statistical Graphics. In *2021 IEEE Visualization Conference (VIS)*. IEEE, New Orleans, LA, USA, 201–205. https://doi.org/10.1109/VIS49827.2021.9623291

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

[34] Brittany Kondo and Christopher Collins. 2014. DimpVis: Exploring Time-varying Information Visualizations by Direct Manipulation. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (Dec. 2014), 2003–2012. https://doi.org/10.1109/TVCG.2014.2346250

[35] Brittany Kondo, Hrim Mehta, and Christopher Collins. 2014. Glidgets: Interactive Glyphs for Exploring Dynamic Graphs. *Proc. of IEEE Conf. on Information Visualization (InfoVis)* (2014).

[36] George Lakoff and Mark Johnson. 2008. *Metaphors We Live By*. University of Chicago Press.

[37] Ricardo Langner, Marc Satkowski, Wolfgang Büschel, and Raimund Dachselt. 2021. MARVIS: Combining Mobile Devices and Augmented Reality for Visual Data Analysis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–17. https://doi.org/10.1145/3411764.3445593

[38] Benjamin Lee, Maxime Cordeil, Arnaud Prouzeau, Bernhard Jenny, and Tim Dwyer. 2022. A Design Space For Data Visualisation Transformations Between 2D And 3D In Mixed-Reality Environments. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–14. https://doi.org/10.1145/3491102.3501859

[39] Bongshin Lee, Tim Dwyer, Dominikus Baur, and Xaquín González Veira. 2018. Watches to Augmented Reality: Devices and Gadgets for Data-Driven Storytelling. In *Data-Driven Storytelling*. AK Peters/CRC Press, 153–168.

[40] Benjamin Lee, Xiaoyun Hu, Maxime Cordeil, Arnaud Prouzeau, Bernhard Jenny, and Tim Dwyer. 2021. Shared Surfaces and Spaces: Collaborative Data Visualisation in a Co-located Immersive Environment. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (Feb. 2021), 1171–1181. https://doi.org/10.1109/TVCG.2020.3030450

[41] Tica Lin, Zhutian Chen, Yalong Yang, Daniele Chiappalupi, Johanna Beyer, and Hanspeter Pfister. 2022. The Quest for Omnioculars: Embedded Visualization for Augmenting Basketball Game Viewing Experiences. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–10. https://doi.org/10.1109/TVCG.2022.3209353

[42] Jiazhou Liu, Arnaud Prouzeau, Barrett Ens, and Tim Dwyer. 2020. Design and Evaluation of Interactive Small Multiples Data Visualisation in Immersive Spaces. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, Atlanta, GA, USA, 588–597. https://doi.org/10.1109/VR46266.2020.00081

[43] K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N. Henry Riche, T. Itoh, W. Stuerzlinger, and B. H. Thomas. 2018. *Immersive Analytics*. Springer International Publishing. https://books.google.com.au/books?id=vaVyDwAAQBAJ

[44] Microsoft. 2021. Mixed Reality UX Elements - Mixed Reality. https://learn.microsoft.com/en-us/windows/mixed-reality/design/app-patterns-landingpage

[45] Microsoft. 2022. MixedRealityToolkit-Unity. https://github.com/microsoft/MixedRealityToolkit-Unity

[46] Mark R. Mine, Frederick P. Brooks, and Carlo H. Sequin. 1997. Moving Objects in Space: Exploiting Proprioception in Virtual-Environment Interaction. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 19–26. https://doi.org/10.1145/258734.258747

[47] Tamara Munzner. 2014. *Visualization Analysis and Design*. CRC Press, Taylor & Francis Group, CRC Press is an imprint of the Taylor & Francis Group, an informa business, Boca Raton.

[48] Patrick Reipschläger and Raimund Dachselt. 2019. DesignAR: Immersive 3D-Modeling Combining Augmented Reality with Interactive Displays. In *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces*. ACM, Daejeon Republic of Korea, 29–41. https://doi.org/10.1145/3343055.3359718

[49] Patrick Reipschlager, Tamara Flemisch, and Raimund Dachselt. 2021. Personal Augmented Reality for Information Visualization on Large Interactive Displays. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (Feb. 2021), 1182–1192. https://doi.org/10.1109/TVCG.2020.3030460

[50] George Robertson, Kim Cameron, Mary Czerwinski, and Daniel Robbins. 2002. Animated Visualization of Multiple Intersecting Hierarchies. *Information Visualization* 1, 1 (March 2002), 50–65. https://doi.org/10.1057/palgrave.ivs.9500002

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

[52] Hans Rosling. 2007. The Best Stats You've Ever Seen | Hans Rosling. https://www.youtube.com/watch?v=hVimVzgtD6w

[53] Puripant Ruchikachorn and Klaus Mueller. 2015. Learning Visualizations by Analogy: Promoting Visual Literacy through Visualization Morphing. *IEEE Transactions on Visualization and Computer Graphics* 21, 9 (Sept. 2015), 1028–1044. https://doi.org/10.1109/TVCG.2015.2413786

[54] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2019. Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. https://doi.org/10.1109/TVCG.2019.2934281

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

[56] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative Interaction Design for Data Visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. ACM, Honolulu Hawaii USA, 669–678. https://doi.org/10.1145/2642918.2647360

[57] B. Shneiderman. 1996. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE Comput. Soc. Press, Boulder, CO, USA, 336–343. https://doi.org/10.1109/VL.1996.545307

[58] Ronell Sicat, Jiabao Li, Junyoung Choi, Maxime Cordeil, Won-Ki Jeong, Benjamin Bach, and Hanspeter Pfister. 2019. DXR: A Toolkit for Building Immersive Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 715–725. https://doi.org/10.1109/TVCG.2018.2865152

[59] Jim Smiley, Benjamin Lee, Siddhant Tandon, Maxime Cordeil, Lonni Besançon, Jarrod Knibbe, Bernhard Jenny, and Tim Dwyer. 2021. The MADE-Axis: A Modular Actuated Device to Embody the Axis of a Data Dimension. *Proceedings of the ACM on Human-Computer Interaction* 5, ISS (Nov. 2021), 1–23. https://doi.org/10.1145/3488546

[60] Dag Svanæs. 2001. Context-Aware Technology: A Phenomenological Perspective. *Human–Computer Interaction* 16, 2-4 (Dec. 2001), 379–400. https://doi.org/10.1207/S15327051HCI16234_17

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

[62] John R Thompson, Zhicheng Liu, and John Stasko. 2021. Data Animator: Authoring Expressive Animated Data Graphics. In *Proceedings of the 2021 CHI*

Conference on Human Factors in Computing Systems. ACM, Yokohama Japan, 1–18. https://doi.org/10.1145/3411764.3445747

[63] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. 2002. Animation: Can It Facilitate? *International Journal of Human-Computer Studies* 57, 4 (Oct. 2002), 247–262. https://doi.org/10.1006/ijhc.2002.1017

[64] J. A. Wagner Filho, C.M.D.S. Freitas, and L. Nedel. 2018. VirtualDesk: A Comfortable and Efficient Immersive Information Visualization Approach. *Computer Graphics Forum* 37, 3 (June 2018), 415–426. https://doi.org/10.1111/cgf.13430

[65] Zhanyong Wan, Walid Taha, and Paul Hudak. 2002. Event-Driven FRP. In *Practical Aspects of Declarative Languages*, Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Shriram Krishnamurthi, and C. R. Ramakrishnan (Eds.). Vol. 2257. Springer Berlin Heidelberg, Berlin, Heidelberg, 155–172. https://doi.org/10.1007/3-540-45587-6_11

[66] Qianwen Wang, Zhen Li, Siwei Fu, Weiwei Cui, and Huamin Qu. 2019. Narvis: Authoring Narrative Slideshows for Introducing Data Visualization Designs. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 779–788. https://doi.org/10.1109/TVCG.2018.2865232

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

[68] Wesley Willett, Yvonne Jansen, and Pierre Dragicevic. 2017. Embedded Data Representations. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (Jan. 2017), 461–470. https://doi.org/10.1109/TVCG.2016.2598608

[69] Amanda Williams, Eric Kabisch, and Paul Dourish. 2005. From Interaction to Participation: Configuring Space Through Embodied Interaction. In *UbiComp 2005: Ubiquitous Computing (Lecture Notes in Computer Science)*, Michael Beigl, Stephen Intille, Jun Rekimoto, and Hideyuki Tokuda (Eds.). Springer, Berlin, Heidelberg, 287–304. https://doi.org/10.1007/11551201_17

[70] Yalong Yang, Tim Dwyer, Kim Marriott, Bernhard Jenny, and Sarah Goodwin. 2020. Tilt Map: Interactive Transitions Between Choropleth Map, Prism Map and Bar Chart in Immersive Environments. *IEEE Transactions on Visualization and Computer Graphics* 27, 12 (2020), 4507–4519. https://doi.org/10.1109/TVCG.2020.3004137

[71] Stephanie Yee and Tony Chu. 2015. A Visual Introduction to Machine Learning. http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

[72] Jonathan Zong, Dylan Wootton, Arvind Satyanarayan, and Josh Pollock. 2022. Animated Vega-Lite: Unifying Animation with a Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–11. https://doi.org/10.1109/TVCG.2022.3209369