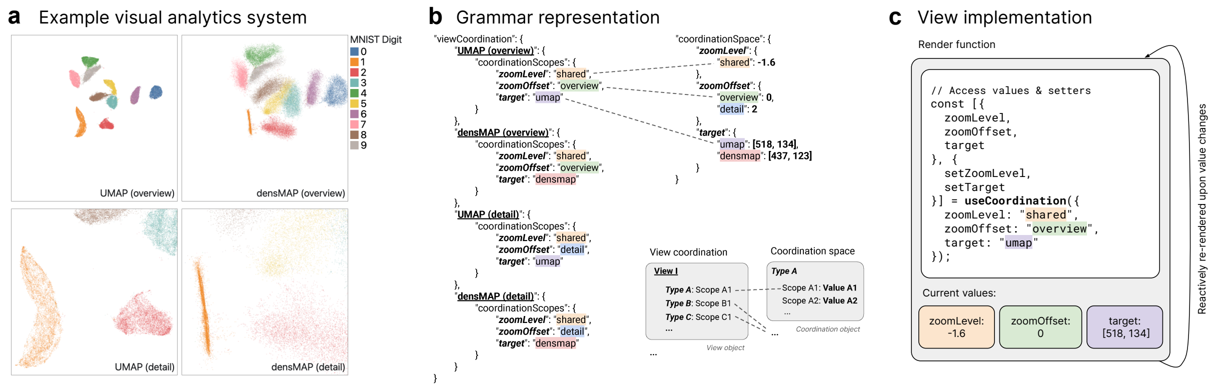
Use-Coordination: Model, Grammar, and Library for Implementation of Coordinated Multiple Views

Mark S. Keller* Harvard Medical School
Trevor Manz† Harvard Medical School
Nils Gehlenborg‡ Harvard Medical School



```
"viewCoordination": {
    "UMAP (overview)": {
        "coordinationScopes": {
            "zoomLevel": "shared",
            "zoomOffset": "overview",
            "target": "umap"
        }
    },
    "densMAP (overview)": {
        "coordinationScopes": {
            "zoomLevel": "shared",
            "zoomOffset": "overview",
            "target": "densmap"
        }
    },
    "UMAP (detail)": {
        "coordinationScopes": {
            "zoomLevel": "shared",
            "zoomOffset": "detail",
            "target": "umap"
        }
    },
    "densMAP (detail)": {
        "coordinationScopes": {
            "zoomLevel": "shared",
            "zoomOffset": "detail",
            "target": "densmap"
        }
    }
}
```

```
"coordinationSpace": {
    "zoomLevel": {
        "shared": -1.6
    },
    "zoomOffset": {
        "overview": 0,
        "detail": 2
    },
    "target": {
        "umap": [518, 134],
        "densmap": [437, 123]
    }
}
```

```
// Access values & setters
const [{
  zoomLevel,
  zoomOffset,
  target
}, {
  setZoomLevel,
  setTarget
}] = useCoordination({
  zoomLevel: "shared",
  zoomOffset: "overview",
  target: "umap"
});
```

Figure 1: Example of a visual analytics system that implements coordinated multiple views with the `use-coordination` software library. a) The system supports comparison of two dimensionality reduction results using four scatterplots linked on subsets of properties. b) The coordination specification defines which views are linked to which properties using a JSON grammar representation. Dotted lines illustrate that the first view "UMAP (overview)" is mapped to particular coordination scopes in the coordination space. c) Views use their coordination scope mappings to obtain values and setter functions from the parent system that manages the coordination specification. Developers can assume that the render function for each view will re-execute upon changes to the values in the mapped coordination scopes.

Abstract

Coordinated multiple views (CMV) in a visual analytics system can help users explore multiple data representations simultaneously with linked interactions. However, the implementation of coordinated multiple views can be challenging. Without standard software libraries, visualization designers need to re-implement CMV during the development of each system. We introduce `use-coordination`, a grammar and software library that supports the efficient implementation of CMV. The grammar defines a JSON-based representation for an abstract coordination model from the information visualization literature. We contribute an optional extension to the model and grammar that allows for hierarchical coordination. Through three use cases, we show that `use-coordination` enables implementation of CMV in systems containing not only basic statistical charts but also more complex visualizations such as medical imaging volumes. We describe six software extensions, including a graphical editor for manipulation of coordination, which showcase the potential to build upon our coordination-focused declarative approach. The software is open-source and available at `https://use-coordination.dev`.

Index Terms: Visualization toolkits, visual analytics, domain specific languages.

*e-mail: mark_keller@hms.harvard.edu
†email: trevor_manz@g.harvard.edu
‡e-mail: nils@hms.harvard.edu

1 Introduction

Coordinated multiple views (CMV) is a visual analytics technique in which interactions in one visualization are automatically reflected in other linked visualizations. CMV is especially powerful during exploratory data analysis as it can help users to discover relationships, make comparisons, and simultaneously view details plus context.

Research on CMV began as early as the 1970s with methods for linking static views [10]. This was followed by a focus on the development of theoretical models which would enable flexible coordination implementations [10, 21, 22, 12]. Several such models were introduced alongside software implementations that are now obsolete. A survey by Roberts *et al.* [23] characterized data structures and underpinning technologies for CMV as an "afterthought."

The translation of theoretical models into shared open-source software implementations has remained under-studied by the visualization research community. As a consequence, visualization tool developers must devise custom implementations each time they want to integrate a technique such as CMV. This places a burden on not only developers, but also on users who may encounter a wide range of *ad hoc* implementations of techniques.

To address this problem, we contribute the `use-coordination` grammar and software library (Fig. 1). Our approach implements the Abstract Model for coordination proposed by Boukhelifa *et al.* [12]. Additionally, we formalize an extension of the model that enables the specification of a hierarchy of coordination. We define the grammar in the form of a JSON-based domain-specific language (DSL) that is declarative and programming language-agnostic. The `use-coordination` library is implemented in JavaScript and can be integrated into React-based web applications, and we provide an alternative implementation for Python. The JavaScript library is accompanied by modules for visualization and editing of coordination specifications using their graph-based representation, for integration with provenance tracking approaches [16], and for construction of specifications imperatively via an object-oriented API. We anticipate `use-coordination` being useful to researchers aiming to support CMV in visual analytics systems by leveraging a common implementation that is grounded in a theoretical model from the information visualization literature. The software is open-source and available with accompanying documentation at `https://github.com/keller-mark/use-coordination`. We demonstrate the utility of the library through use cases with statistical plots and medical imaging visualizations.

2 Design Goals

We had several design goals based on our experiences implementing CMV in visual analytics tools for biomedical data [19, 15]:

DG1. Declarative grammar representation. Declarative representations of state enable serialization for use cases such as database storage, programming language portability, reproducibility, provenance tracking, and multiplayer interactions. We aim for a coordination model that has a declarative representation which can be serialized to a JSON-based DSL format.

DG2. Reactive updates. Today's most popular user interface development frameworks are designed for reactive component-based architectures. In these architectures, a tree of view components re-render reactively, in response to changes in state values. The reactive paradigm simplifies the mental model for developers who do not need to worry about component state becoming stale thanks to efficient component tree diffing algorithms that are abstracted by the framework. In JavaScript, this class of frameworks includes React [2], Vue [8], Svelte [6], and Solid [5]. The reactive architecture contrasts with earlier publish-subscribe architectures that rely on explicit event subscriptions. Integrating event-based state management approaches into reactive frameworks can be unnatural, forcing the developer to revert to a more complicated mental model in which there is a risk of component state becoming stale (*e.g.*, due to accidental lack of publication or subscription). We aim for a coordination model and library that are designed for reactive component rendering architectures. As a corollary, view implementations should remain independent of one another such that views do not need to communicate directly to achieve coordination.

DG3. Decoupled from data representation and view layout. For maximum flexibility, coordination should be decoupled from the processes of data representation (and by extension, loading and transformation) and view layout. Not only does this allow developers to use methods of their choice (*i.e.*, view layout via CSS or a specialized JavaScript library) and visualize diverse data types (*e.g.*, tables, images, trees, graphs), it also follows the Unix philosophy of modular software tools that have a singular focus.

DG4. Arbitrary coordination types. Coordination should not be limited to a predefined set of properties. Developers should be able to coordinate any part of the state of an application for maximum flexibility.

3 Related Work

CMV techniques have been studied by many visualization researchers over the years. We refer to Roberts [23] and Scherr [25] for two early surveys.

3.1 Coordination Models

Roberts [23] describes multiple model architectures for coordination, including data-centric and model-view-controller (MVC) architectures. Data-centric approaches are powerful but to our knowledge all existing approaches in this category are limited to tabular data representations. In contrast, Boukhelifa *et al*. [12] contribute the Abstract Model for coordination which is not data-centric and therefore more general than prior approaches. Multiple visualization systems, including CViews [13] and ManiVault [28], implement this model, but there is not a standalone implementation nor a standard declarative representation for the model state. Further, existing implementations of CMV use event-based approaches for information propagation, which are unnatural to use with reactive user interface frameworks such as React.

3.2 Grammars for Coordination

Harth *et al*. [18] and Chen *et al*. [14] both describe the goal of enabling CMV across views implemented using different plotting libraries using JSON- and natural language- based grammars, respectively. Both approaches are event-based with predefined sets of supported actions and both make assumptions about tabular data types. Neither encodes key rudiments of the Abstract Model [12] such as coordination types and coordination scopes. Though both are open-source, neither is available as a documented reusable software library through a package repository such as NPM.

3.3 Interaction Orchestration

Interaction orchestration can be considered as a special case of CMV in which all coordination types represent interactions such as selections and brushing. The Vega-Lite Grammar of Interaction [24] supports declarative specification of coordinated selections comprising points and intervals. The Grammar of Interaction is powerful but remains limited to the scope of a Vega-Lite specification (*i.e.*, unable to orchestrate across multiple specifications or outside Vega-Lite), does not support arbitrary value schema, and supports only single- or all-view coordination. The Dynamically Interactive Visualization (DIVI) [26] approach infers configurations for coordinated interactions from views defined as SVGs. The usage of SVG enables implementation of CMV with minimal developer overhead, but comes with scalability limitations and precludes coordination of non-SVG views such as tables or input elements. Both the Vega-Lite and DIVI approaches are implemented using event-based models.

3.4 Data Flow Models

A number of data flow approaches for coordinated visualization have been proposed, including VisFlow [29] and sMolBoxes [27]. Data flow systems visually encode operations on data, helping users to understand which views are linked on which data elements and rendering properties. In contrast, more general visual analytics systems, even those which implement CMV, do not typically include such a visual encoding. Similar to data-centric coordination models, data flow models make assumptions about data representations (*e.g.*, that data is tabular). To our knowledge, there are not standard declarative representations for data flow models. We note that research on visualization for data flow models is complementary to research on coordination and may inform how coordination model state is communicated to users.

4 The Use-Coordination Model and Grammar

The `use-coordination` model can be seen as an extension of the Abstract Model contributed by Boukhelifa *et al*. [12, 13]. The model has also been informed by our earlier usage of the Abstract Model to implement CMV in the Vitessce visual analytics framework [19]. We formalize the model as a JSON-based grammar in terms of each rudiment of coordination:

Coordination Type. A coordination type is a name for a property being coordinated (e.g., `zoom` or `colormap`).

Coordination Scope. A coordination scope is a named instance of a coordination type. For example, a visual analytics system with two different scatterplots displaying different projection results may have two coordination scopes for the `zoom` coordination type, `zoom.UMAP` and `zoom.TSNE`, enabling the user to zoom in these plots independently. We do not associate any semantics with coordination scope names, although particular visual analytics systems or view implementations may.

Coordination Value. Each coordination scope is mapped to a value, for example `zoom.UMAP: 10`.

Coordination Object. A coordination object encapsulates the coordination scopes and values for a given coordination type. A coordination object may contain zero or more coordination scopes. Coordination scope names must be unique within a coordination object but may be reused across coordination objects.

Coordination Space. The coordination space contains the coordination objects for all coordination types in the system (Fig. 1b).

View Coordination. Views typically correspond to plots or other standalone widgets in a visual analytics system. Each view may have a mapping from coordination types to coordination scopes (Fig. 1). Depending on their implementation, views may support a subset of the coordination types that are present in the coordination space. During rendering, each view performs a lookup (in the coordination space) for the coordination values that correspond to its current coordination scope mappings.

Hierarchy of Coordination. Prior work has discussed the need for hierarchical coordination, though did not formalize a multi-level model. Boukhelifa and Rodgers note, "...theoretically, any process could be coordinated with anything," [13] and McDonald *et al*. state, "The hierarchical or graph structure of the Scene will often reflect a similar hierarchy in the Subjects," [21]. Our grammar extends the Abstract Model [12] to support a hierarchy of coordination. First, we allow each view to be associated with more than one scope per coordination type (*i.e.*, an array of scopes). Next, a `coordinationScopesBy` property can be specified for each view, which defines scope $\rightarrow$ scope mappings, including between scopes within different coordination objects. For example, scopes representing bars selected in a bar plot can be mapped to scopes representing colors, enabling users to define color encodings per selected bar. To our knowledge, such an extension of the Abstract Model [12, 13] has not previously been formalized. Further details about the JSON representation for hierarchies of coordination are available in Section 2 of our Supplemental Materials.

In accordance with our design goals, intentionally absent from the grammar are events (DG2), layout- and data- related properties (DG3), and view-view mappings (DG5).

5 Software Implementation

We contribute an open-source software library called `use-coordination`, which implements the model and grammar defined in Section 4. The library follows the React "hooks" paradigm in which composable functions abstract away complex state management features. Developers of visual analytics systems can use the library by defining a *coordination specification* (*i.e.*, an instance of the grammar) and defining each view as a React component instance that is associated with a unique identifier. Within each view, developers can use the provided hook functions to perform lookups of coordination values from the coordination space and to obtain "setter" functions that enable updating the values in the mapped coordination scopes.

Software Extensions

The portability and declarative nature of the `use-coordination` grammar have proven to be powerful in enabling the straightforward development of additional software modules (Fig. 2).

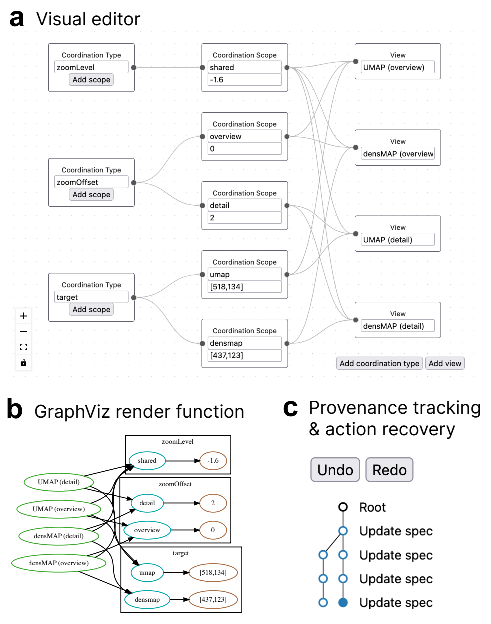


Figure 2: Extensions of `use-coordination`. a) Our visual editor enables users to modify the coordination specification without knowledge of JSON. b) The coordination specification can be rendered to the GraphViz DOT format for static visualization. c) User interactions in a visual analytic system can be tracked through optional integration with Trrack [16].

Python and non-React implementations. Our approach and JSON representation generalize beyond React and JavaScript. First, we demonstrate an implementation of the `useCoordination` hook function for Solid [5] using its *signals* pattern for reactivity[1]. Second, we provide `with-coordination`, a Python implementation that coordinates Jupyter widget views[2] . Importantly, the software modules that follow in this section are designed around the shared JSON grammar, such that each module is compatible with all implementations.

Programmatic specification. On top of the declarative JSON representation, we have developed an object-oriented API for incremental construction and manipulation of coordination specifications. This API enables users to define objects corresponding to views and coordination scopes. Subsequently, users can pass these objects to functions such as `linkViews`, avoiding errors associated with direct JSON manipulation. At any point, users can convert their specification object to JSON.

Node-link diagrams. The `use-coordination` model takes the form of a graph data structure, which can be visualized as a node-link diagram. We provide a function to convert coordination specifications from JSON to GraphViz [17] `DOT` format for visualization (Fig. 2b).

Visual editor. We provide a visual flow-based editor based on the graph structure of the model. This editor is implemented as a React component using the ReactFlow [4] library, enabling it to be embedded into visual analytics applications so that end-users can modify the coordination specification *in situ* (Fig. 2a).

Provenance tracking. The serializable nature of the JSON-based coordination specification lends itself to integration with

[1]https://github.com/keller-mark/use-coordination-solid
[2]https://github.com/manzt/with-coordination

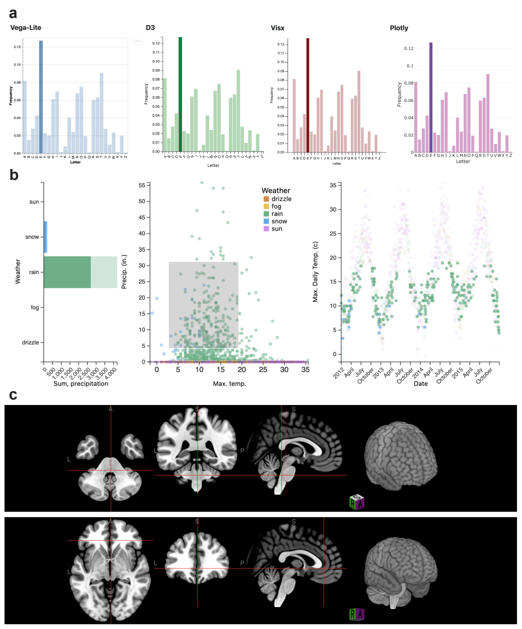


Figure 3: Use cases. a) Coordinated selections across Vega-Lite, D3, Visx, and Plotly bar plot implementations. b) Seattle weather example in which brushing interactions in one plot (the middle scatterplot) are coordinated with the other two plots. c) Coordination of 2D (crosshair position) and 3D (azimuth) parameters of a medical imaging volume.

provenance tracking and action recovery approaches. We provide a utility function to help integrate `use-coordination` with Trrack [16] (Fig. 2c).

Schema validation. Coordination types can optionally be associated with schema to which their values must conform. We implement a React provider component which supports validation via Zod [9] schema. Schema validation errors can be caught and handled by the parent application to display friendly error messages and prevent invalid data from propagating to views.

6 Use Cases

We demonstrate `use-coordination` through three use cases. Based on our experiences using the library, we also provide detailed usage guidance in Section 3 of our Supplemental Materials.

6.1 Use Case 1: Multiple Plotting Libraries

Visual analytics systems may integrate multiple plotting libraries. There is often a need to link elements across plots regardless of how they are implemented, a use case which `use-coordination` supports. We implement a bar plot using four different plotting libraries: Vega-Lite [24], D3 [11], Visx [7], and Plotly [3] (Fig. 3a). With `use-coordination`, we coordinate bar selections across all four plots. The user can dynamically modify which subsets of plots are coordinated together via our visual editor extension (Fig. 2).

6.2 Use Case 2: Seattle Weather

We demonstrate that `use-coordination` can be used to link brushing interactions across plots. We reproduce Figure 1 of Snyder and Heer [26] in which the Seattle weather dataset is visualized using a bar plot and two scatterplots (Fig. 3b). As the user brushes (in the middle scatterplot view) a region containing low temperature and high precipitation, the data points are highlighted in all three plots.

6.3 Use Case 3: Medical Imaging

Our approach is not limited to coordination of visualizations of tabular data. We visualize a medical imaging volume using two Niivue [1] instances (Fig. 3c). We link one of three 2D parameters (X-axis crosshair) and one of two 3D parameters (azimuth) via `use-coordination`. This coordination enables the user to simultaneously visualize six 2D slices (two of which are identical) and view the volume from two different angles.

7 Discussion

With `use-coordination`, we contribute the concept of a declarative coordination specification that is decoupled from both data representation and view layout. We provide a software ecosystem around such specifications oriented towards visual analytics system developers. The coordination specification is grounded in an established coordination model [12] whose influence on prior works in the visualization field demonstrates wide applicability.

The most closely related works are from Harth *et al*. [18] and Chen *et al*. [14], yet both differ from `use-coordination` in terms of design goals and conceptual models. These works are based on event- or action-based models which stand in contrast to the declarative and reactive approach that we adopt (DG1, DG2). In comparison to Harth *et al*., our work also differs in that it is decoupled from data representation and view layout (DG3). Compared to the natural language-based grammar of Chen *et al*., we developed a declarative JSON-based DSL (DG1). This declarative approach makes it easy to implement multiple systems around the DSL. For instance, our grammar has the potential to serve as a compilation target for natural language-based approaches involving large language models.

The simplicity of our approach comes with trade-offs. For example, we allow for coordination types to have arbitrary names and value schema (DG4) and we decouple coordination from view layout (DG3). While this freedom enables the integration of `use-coordination` with any selection, interaction, and layout approaches, it could present challenges to reuse of coordination specifications across visualization system boundaries or to meta-analyses that rely on standardization of coordination type names with associated semantics. Despite this limitation, we emphasize that visualization developers could define their own set of standard coordination types with domain-specific semantics (*e.g.*, `biomarker-selection` for biological data visualization). Our grammar does not explicitly include aspects such as transitions/easing or composite brushing [20], which would be up to the developer to implement (coordinated or otherwise). We note that developers may define more fully-featured libraries on top of `use-coordination` that are coupled to particular view layout approaches or coordination type semantics.

8 Conclusion

In this work, we present the `use-coordination` model, grammar, and software library. We demonstrate that standardizing the representation of coordination through a declarative JSON-based grammar enables the development of software extensions that simplify the implementation and reasoning about CMV. Our use of reactive software paradigms and initial extensions (Sec. 5) reduce redundant work and lower barriers to entry across multiple platforms and languages. An important question for future work is how to effectively communicate coordination to end-users. We anticipate our work serving as a tool to address this and other human-centered questions.

`Use-coordination` is open-source and our software packages are available through the NPM and PyPI package repositories. The software is accompanied by tests, documentation, and examples.

Supplemental Materials

All supplemental materials are available on OSF at https://doi.org/10.17605/OSF.IO/SEJN5, released under a CC BY 4.0 license. Our supplemental materials include further details about the JSON grammar and the model that we have developed to represent hierarchies of coordination.

Acknowledgments

This research was supported by the National Institutes of Health (OT2OD033758, T15LM007092, T32HG002295, R33CA263666) and the National Science Foundation (DGE1745303).

References

[1] Niivue. https://github.com/niivue/niivue, 2024. Last accessed 2024-07-25. 4

[2] React. https://github.com/facebook/react, 2024. Last accessed 2024-07-25. 2

[3] react-plotly.js. https://github.com/plotly/react-plotly.js, 2024. Last accessed 2024-07-25. 4

[4] ReactFlow. https://github.com/xyflow/xyflow, 2024. Last accessed 2024-07-25. 3

[5] SolidJS. https://github.com/solidjs/solid, 2024. Last accessed 2024-07-25. 2, 3

[6] Svelte. https://github.com/sveltejs/svelte, 2024. Last accessed 2024-07-25. 2

[7] Visx. https://github.com/airbnb/visx, 2024. Last accessed 2024-07-25. 4

[8] Vue. https://github.com/vuejs/core, 2024. Last accessed 2024-07-25. 2

[9] Zod. https://github.com/colinhacks/zod, 2024. Last accessed 2024-07-25. 4

[10] R. A. Becker, W. S. Cleveland, and A. R. Wilks. Dynamic Graphics for Data Analysis. *Statistical Science*, 2(4):355–383, Nov. 1987. doi: 10.1214/ss/1177013104 1

[11] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Oct. 2011. doi: 10.1109/TVCG.2011.185 4

[12] N. Boukhelifa, J. C. Roberts, and P. J. Rodgers. A coordination model for exploratory multiview visualization. In *Proc. CMV*, pp. 76–85. IEEE, London, England, 2003. doi: 10.1109/CMV.2003.1215005 1, 2, 3, 4

[13] N. Boukhelifa and P. J. Rodgers. A model and software system for coordinated and multiple views in exploratory visualization. *Information Visualization*, 2(4):258–269, Dec. 2003. doi: 10.1057/palgrave.ivs.9500057 2, 3

[14] R. Chen, X. Shu, J. Chen, D. Weng, J. Tang, S. Fu, and Y. Wu. Nebula: A coordinating grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4127–4140, Apr. 2022. doi: 10.1109/TVCG.2021.3076222 2, 4

[15] F. Cheng, M. S. Keller, H. Qu, N. Gehlenborg, and Q. Wang. Polyphony: an interactive transfer learning framework for single-cell data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):591–601, Jan. 2023. doi: 10.1109/TVCG.2022.3209408 2

[16] Z. Cutler, K. Gadhave, and A. Lex. Trrack: A library for provenance-tracking in web-based visualizations. In *Proc. VIS*, pp. 116–120, Oct. 2020. doi: 10.1109/VIS47514.2020.00030 2, 3, 4

[17] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. *Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools*, pp. 127–148. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-642-18638-7_6 3

[18] P. Harth, A. Bast, J. Troidl, B. Meulemeester, H. Pfister, J. Beyer, M. Oberlaender, H.-C. Hege, and D. Baum. Rapid prototyping for coordinated views of multi-scale spatial and abstract data: A grammar-based approach. In *Proc. VCBM*, 2023. doi: 10.2312/vcbm.20231218 2, 4

[19] M. S. Keller, I. Gold, C. McCallum, T. Manz, P. V. Kharchenko, and N. Gehlenborg. Vitessce: a framework for integrative visualization of multi-modal and spatially-resolved single-cell data. Technical report, OSF Preprints, Oct. 2021. doi: 10.31219/osf.io/y8thv 2

[20] A. R. Martin and M. O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proc. VIS*, pp. 271–278, Oct. 1995. doi: 10.1109/VISUAL.1995.485139 4

[21] J. A. McDonald, W. Stuetzle, and A. Buja. Painting multiple views of complex objects. *ACM SIGPLAN Notices*, 25(10):245–257, Sept. 1990. doi: 10.1145/97946.97975 1, 3

[22] C. North and B. Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proc. AVI*, pp. 128–135. Association for Computing Machinery, New York, NY, USA, 2000. doi: 10.1145/345513.345282 1

[23] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proc. CMV*, pp. 61–71. IEEE, Zurich, Switzerland, 2007. doi: 10.1109/CMV.2007.20 1, 2

[24] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, Aug. 2017. doi: 10.1109/TVCG.2016.2599030 2, 4

[25] M. Scherr. Multiple and coordinated views in information visualization. Technical report, Trends in information visualization, 2008. 2

[26] L. S. Snyder and J. Heer. DIVI: Dynamically Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):403–413, Jan. 2024. doi: 10.1109/TVCG.2023.3327172 2, 4

[27] P. Ulbrich, M. Waldner, K. Furmanova, S. M. Marques, D. Bednar, B. Kozlikova, and J. Byska. sMolBoxes: Dataflow model for molecular dynamics exploration. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):581–590, Sept. 2023. doi: 10.1109/TVCG.2022.3209411 2

[28] A. Vieth, T. Kroes, J. Thijssen, B. van Lew, J. Eggermont, S. Basu, E. Eisemann, A. Vilanova, T. Hollt, and B. Lelieveldt. ManiVault: A flexible and extensible visual analytics framework for high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 30(01):175–185, Jan. 2024. doi: 10.1109/TVCG.2023.3326582 2

[29] B. Yu and C. T. Silva. VisFlow - web-based visualization framework for tabular data with a subset flow model. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):251–260, Jan. 2017. doi: 10.1109/TVCG.2016.2598497 2