EasyPZ.js: Interaction Binding For Pan and Zoom Visualizations

Michail Schwab* Khoury College of Computer Sciences Northeastern University
James Tompkin† Brown University
Jeff Huang‡ Brown University
Michelle A. Borkin§ Khoury College of Computer Sciences Northeastern University

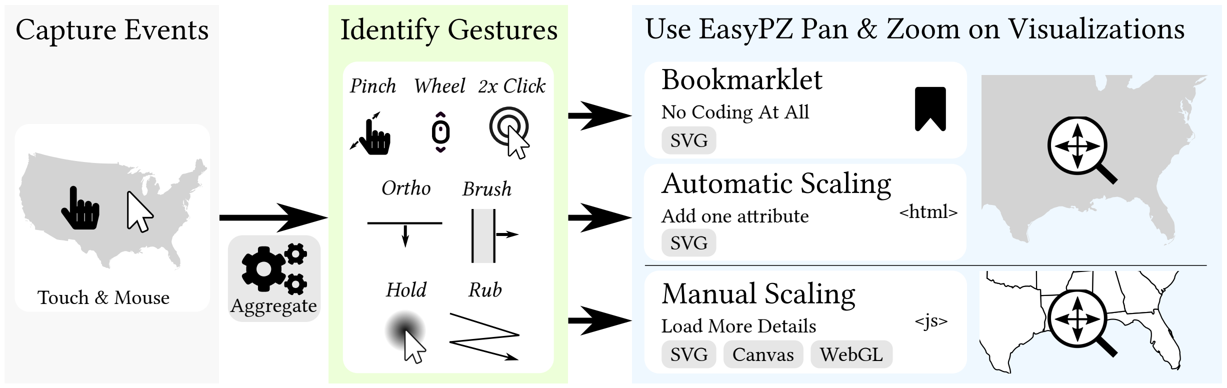


Figure 1: EasyPZ captures mouse and touch events, aggregates the events, and identifies pan and zoom gestures, such as pinch or brush zoom. These gestures can be used for pan and zoom in three different ways: 1. The EasyPZ bookmarklet temporarily enables pan and zoom SVG visualizations without any code changes. 2. The automatic mode of EasyPZ permanently enables pan and zoom on a SVG visualization with a simple HTML change. 3. In the manual EasyPZ mode, a SVG, canvas, or WebGL visualization can respond to pan and zoom by providing detail on demand, such as showing state borders when zooming in to a country.

Abstract

The creation of data visualizations has become easier as the skill-barrier to our tools has decreased. However, adding interactivity, such as gestures for pan and zoom, still requires significant coding expertise. We introduce an open-source library—EasyPZ.js—for the creation of multi-scale (pan and zoom) visualizations across desktop and mobile devices. EasyPZ is fully customizable and extendable with flexible options for interaction design. For example, it is easy to choose gestures which are compatible with selection interactions such as clicking. EasyPZ can be enabled on any SVG-based visualization on the web with one line of code, or by simply clicking a bookmark without requiring commitment to code changes. With this library, we contribute ways for the visualization community to more easily author interactive multi-scale visualizations.

Keywords: Human-computer interaction, SVG, visualization, interaction binding, pan, zoom, navigation, toolkit.

1 Introduction

In recent years, many tools have been published in the data visualization community to simplify the creation process of custom data visualizations (e.g., [5, 19]). Some tools even allow the creation of complex data visualizations without any coding at all [10, 12]. However, the availability of simple-to-use interaction techniques for these visualizations has not kept pace: even enabling the default pan and zoom in D3.js requires at least 6 lines of code.

Pan and zoom interactions are critical as they allow us to explore multi-scale data on fixed-size displays [7]. As large datasets increase in prevalence, visualization creators will increasingly be asked to consider how best to design both visualization appearance and interaction behavior, especially for mobile devices, where small screens pose navigation challenges for data visualizations.

For interaction, there are many implementations for pan (scroll), zoom in, and zoom out operations, with different compatibilities across mobile and desktop. Different interaction designs can have dramatic effects on performance [18], able to turn an interface from one that frustrates users to one that supports them. While the HCI community produces novel interaction designs for pan and zoom, e.g., [3, 15], the visualization community benefits less from these findings due to opaque or closed-source implementations. The lack of an open library for such interactions makes it harder for techniques to bridge between the fields, and for more efficient research techniques to be applied in software generally.

To address these challenges, we introduce an extensible open-source pan and zoom library to create interactive multi-scale data visualizations across mobile and desktop devices. EasyPZ.js can be enabled on most data visualizations on the Web by simply clicking a fully customizable bookmarklet, or by adding an "easypz" HTML attribute to an SVG element. EasyPZ offers a simple way to test, tune, and use interaction techniques for multi-scale visualizations.

*email: michaschwab@ccs.neu.edu
†email: james_tompkin@brown.edu
‡email: jeff_huang@brown.edu
§email: m.borkin@neu.edu

2 Related Work

2.1 Multi-scale Interaction Techniques

A large body of work exists on interaction techniques for pan and zoom, and new techniques are frequently proposed. Early multi-scale or zoomable interfaces, such as Pad, Pad++, and MuSE [4, 8, 16] defined zoom based on holding a mouse button down or dragging a slider. The interaction design space has been explored through brush or marquee zoom, anchor or ortho zoom [1, 3], pinch to zoom [11], elastic interaction for precise manipulation [14], pan-speed-dependent zooming [9], flick pan or zoom [17], and cyclic gestures based on rubbing [15] or circles [13].

While many techniques exist, there is little overview to inform data visualization creators about which techniques to use. This lack of overview and accessibility of techniques can in part be attributed to the difficulty of contributing a new technique to a larger framework during or after the development process. Since there is no standardized way to create, combine, tune, distribute and share techniques, they are often implemented in isolation with closed source, which makes reproduction and comparison difficult. This can hinder adoption, which in turn slows progress because techniques are not being used or systematically compared.

2.2 Data Visualization Enhancer Libraries

Some works create JavaScript libraries which can be added to existing data visualizations to enhance their functionality. In particular, VisDock [6] has enabled a large set of features on existing data visualizations, such as advanced selection techniques and even pan and zoom. However, VisDock cannot be injected into existing visualizations automatically and is not designed to work seamlessly with the existing visualization, but instead adds a predefined panel with these functions. This can be useful, but limits the data visualization creator's ability to freely design the user interface.

2.3 Pan and Zoom Libraries for Data Visualization

Implementing multi-scale interaction is sometimes a tricky and nuanced affair, and so some web-based libraries exist to ease pan and zoom interaction implementation. D3.js [5] aids the implementation of basic pan and zoom functionality on a selection of nodes with cross-platform compatibility and the ability to constrain the amount of panning. Primarily for touch on mobile, Hammer.JS [2] provides access to gestures such as panning, zooming, rotating and swiping, although, the transformation must be implemented manually given the input. Both approaches require manual coding in JavaScript to implement different techniques, with few interaction methods, fixed parameters, and limited extensibility.

3 EasyPZ Library

We contribute EasyPZ.js, an open-source pan and zoom JavaScript library for mobile and desktop computers. It comprises an interchangeable set of 32 pan and zoom interaction techniques with a consistent underlying parameterization, where each exposes configurable parameters. EasyPZ is independent of any visualization and can be used on arbitrary 1D sliders, e.g., to control very long videos, on timelines, or on arbitrary 2D spaces, e.g., for pan and zoom on maps. EasyPZ is compatible with SVG (e.g., D3.js), Canvas, and WebGL (2D); in general, it is compatible with any Web environment in which scale and translation can be set.

3.1 Design Considerations and Features

Drawing from Section 2, the main challenges faced in multi-scale interaction are that (1) using different pan and zoom techniques or varying their parameters is difficult, (2) creating useful techniques is challenging, (3) there is insufficient overview or systematic evaluation, and (4) many data visualization creators only share static versions of their SVG visualizations online (e.g., reddit). To address these challenges, we declare the following system requirements:

Ease of Use [Ease]: An easy to use system simplifies access and use of pan and zoom techniques *(1)*, simplifies the process of using existing techniques in data visualizations *(2)*, and helps visualization creators that have scalable visualizations but do not have the skill or time to implement them in a multi-scale environment *(4)*.

To ease use for non-developers, EasyPZ provides a set of pan and zoom techniques including newly-created techniques. These can enhance an existing visualization via a bookmarklet (i.e., no code changes - see "Bookmarklet" in Fig. 1), which dynamically 'injects' EasyPZ into most SVG-based data visualizations on the Web. This bookmarklet can be customized via a Webpage to use one's favorite interaction methods and parameters, e.g., to address accessibility needs such as slowing down zoom speed in case of lower motor control. Automatically translating and scaling visualization content is possible on SVG visualizations, as they are vector-based, but not on canvas or WebGL visualizations, as they are pixel-based and manipulations would introduce blur. Hence only manual scaling is compatible with all visualization drawing surface types (see Fig. 1).

To ease development, EasyPZ simplifies low-level event handling complexity by providing utility functions such as FRICTIONINTERACTION() and automatically applying computed transformations to data visualizations while respecting pre-existing transformations of elements. Techniques created with EasyPZ can be distributed as a single file which extends EasyPZ for others to use.

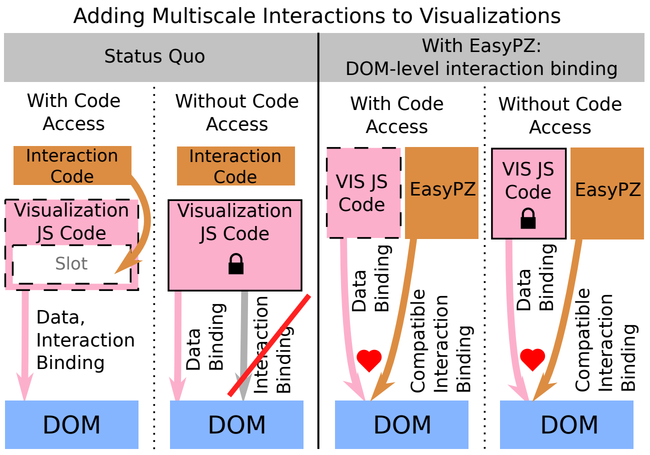


Figure 2: Adding multi-scale interactions to visualizations with and without EasyPZ's DOM-level interaction binding. *Left:* In the status quo, interactions must be manually implemented within the visualization code, which does not work if the code is not available, e.g., Web visualizations or SVG-based visualizations without code. *Right:* EasyPZ's DOM-level interaction binding allows interaction without access to the visualization code, while ensuring compatibility (❤).

Modular and Extensible [ModExt]: Modular techniques can be shared, which simplifies access to techniques *(1)* and makes it easier to create an overview of techniques, for example by simply listing each technique's name, and to evaluate them *(3)*. Creating a modular technique for an extensible platform is also easier than manual implementation *(2)*. Better techniques facilitate exploration of multi-scale data *(4)*. EasyPZ is implemented in a modular way.

Standardization [Standard]: A successful system should standardize pan and zoom techniques so systematic evaluations can be completed in a repeatable way *(3)*. EasyPZ standardizes parameters across techniques (e.g., Schwab et al. [18]). This makes it easy to tune interactions ([**Tune**]) and their parameters as techniques can be swapped in and out on-the-fly, e.g. to enable faster zooming, or to handle physical impairments such as inability to use the mouse wheel or to click twice within some time frame. This standardization allows EasyPZ to help with finding and tuning interactions which are compatible: both self-compatible and compatible with existing visualization interactions such as selection. By swapping in and out individual techniques, their sometimes obscure compatibilities become clear. Further, techniques are often designed with a specific user group or set of tasks in mind, e.g., beginner or expert users may have different requirements. EasyPZ allows such requirements to be tested on-the-fly, and exported to data visualizations immediately.

3.2 Implementation

EasyPZ consists of three parts: The **extensions**, **core** library, and the **loader**. The extensions make up all pan and zoom interaction techniques by declaring names, parameters and event listeners, and returning the change in transformation that an event, such as mouse move, has caused. The core library keeps track of the total transformation, sends enabled extensions the latest data, such as current and previous mouse positions, and provides utility functions to simplify momentum-based interactions or to maximize compatibility. The loader uses this total transformation to apply it to any EasyPZ-enabled element, be it enabled via JavaScript, via bookmarklet, or HTML, and does this in a manner that respects pre-existing transformations by computing a combined transformation.

Fig. 2 shows the difference between traditional interaction implementation and EasyPZ's DOM-level interaction binding. Traditionally, adding multi-scale interactions to visualizations requires manual work to add an interaction implementation to the visualization's code such that the visualization's code base handles both the data binding as well as the interaction binding. Depending on the visualization's structure and existing transformations, this can be challenging, and is impossible if there is no access to a code base, e.g., for visualizations in the Web, or for static SVG-based visualizations that do not have corresponding JavaScript code.

EasyPZ solves this problem by bypassing the visualization's code. Where the visualization's code handles data binding using D3.js or another library, EasyPZ handles the interaction binding by accessing the DOM directly while respecting existing transformations.

3.3 Technical Design Choices

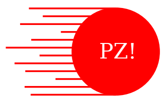
Additive and Multiplicative Transformations: If top level SVG elements already use a transformation as part of the visualization, then this would clash with the dynamic transformation for pan and zoom. EasyPZ applies its transformations on top of any existing SVG transformations, e.g., multiplicative with existing scale transforms and additive to existing translation transforms. This works for any ongoing existing transformations, too: EasyPZ is compatible with visualizations that actively change transformations of elements, such as animated tree maps. EasyPZ does not insert additional elements for simplifying the transformation operation to not change the structure of the SVG, which could break existing selectors. This makes adding pan and zoom easy [Ease].

Conservative Activation: To capture all possible pan events, a pan implementation could wait for a mouse down event, activate, and then listen for mouse move events until the mouse key is released. Unfortunately, such an implementation would be incompatible with many other interactions. For example, *hold zoom* would also trigger this pan implementation, and users would pan and zoom at the same time when trying to zoom by holding down the mouse and moving the mouse for zoom target adjustments. EasyPZ tries to be as compatible as possible with existing interactions and never prevents JavaScript events to be propagated. This is facilitated by EasyPZ's utility functions, such as CALLBACKAFTERTIMEOUTORMOVEMENT, which allows specifying a minimum or maximum pointer travel distance within a time frame to determine whether to react to a possible gesture. For example, *hold zoom* triggers only if the pointer is not moved more than 3 pixels in the first 300 ms of the gesture, whereas *drag pan* requires a minimum movement of 3 pixels in the first 300 ms of the gesture. This enables these two techniques to be compatible with one another, whereas otherwise they would both trigger at the same time and cause unintuitive behavior. Still, some techniques are simply incompatible because they use the same input gesture, such as *drag pan* and *brush zoom*.

3.4 Using EasyPZ with SVG

To add zoom to an existing visualization, its top-level SVG elements need to be transformed. To do this, other libraries require visualization creators to specify a function to be executed when users zoom. In this function, visualization creators would assign the current transformation to the top level elements of their visualization. With EasyPZ, adding pan and zoom interactions to SVG visualizations, like those built with D3.js, does not require any JavaScript code [Ease]. EasyPZ achieves this by automatically applying the transformation to the top level SVG elements by default. Pan and zoom can be added with an HTML attribute in the SVG tag. In Listing 1, EasyPZ controls the top-level transformation of the SVG, and allows us to pan and zoom a circle.

```
<svg easypz>
  <circle cx="100" cy="70"
    ↪  r="25" fill="#f00"></circle>
  <text x="100" y="75">PZ!</text>
</svg>
```



Listing 1: EasyPZ pan and zoom circle, with motion depicted as horizontal lines. jsFiddle at `https://tinyurl.com/easypz-pz`.

By default, EasyPZ applies the transformation to the top level elements in the SVG using the CSS element selector "svg > *". Any valid selector can be used in the "applyTransformTo" attribute, including element classes. In this way it is possible to exclude certain elements, such as legends, and to let EasyPZ apply the transformations only to the chosen elements.

Modular: EasyPZ's technique implementations are fully modular, which helps standardization, sharing of techniques, and extensibility of the library [Standard, ModExt]. By default, EasyPZ enables a common set of compatible pan and zoom interactors: we assign *drag pan*, *hold zoom in*, *click hold zoom out*, *wheel zoom*, *pinch zoom*, *double click zoom in*, and *double right click zoom out*. However, users can 'mix and match' interaction techniques. For example, we might want to enable *flick pan* with *press-and-hold zoom in* and *double-click zoom out* (Listing 2). Because of this interchangeability, visualization creators can choose between techniques that are compatible with other interactions on the website, and swap the activated techniques without any other changes necessary.

```
<svg easypz='{"modes": ["FLICK_PAN",
 ↪  "HOLD_ZOOM_IN", "DBLCLICK_ZOOM_OUT"]}>
   <!--- Visualization --->
</svg>
```

Listing 2: Modular interactors with configurable parameters.

Customizable: Many interactors are customizable [Standard, Tune]. Interactions can also be scale and bounds limited so that users do not become lost, shown in Listing 4.

```
<svg easypz='{"options": { "minScale":
 ↪  0.8, "maxScale": 10, "bounds": { "top": -50,
 ↪  "right": 50, "bottom": 50, "left": -50 }} }'>
   <!--- Visualization --->
</svg>
```

Listing 3: Setting scale and translation bounds.

Progressive Loading: Through EasyPZ's "onTransformed" function, visualizations can react to transformation changes. For example, as indicated in Fig. 1, more detailed map data can be loaded when a user zooms in:

```
<svg easypz='{"onTransformed": "updateVis"}'>
   <!--- Visualization --->
</svg>
<script>
function updateVis(transform) {
   if(transform.scale < 0.8) {
      // Load more data, such as state lines.
   }
   // Rescale SVG
}
</script>
```

Listing 4: Setting scale and translation bounds.

Manual Implementation with JavaScript, D3.js, and canvas Advanced visualizations may wish to react to zooming by re-rendering paths in more detail, update axes and scales, or change map projections. In these cases, more control over the zoom behavior is needed. To address this need, EasyPZ provides access to an "onTransformed" function, which is called when the user zoomed or panned. This is shown in Listing 5.

```
new EasyPZ(visEl, function(transform) {
   // Use transform.scale, .translateX,
    ↪  .translateY to update your visualization.
});
```

Listing 5: Manual EasyPZ Pan and Zoom, e.g. for D3.js or canvas use. The callback function is called whenever a user panned or zoomed using any of the interaction methods specified.

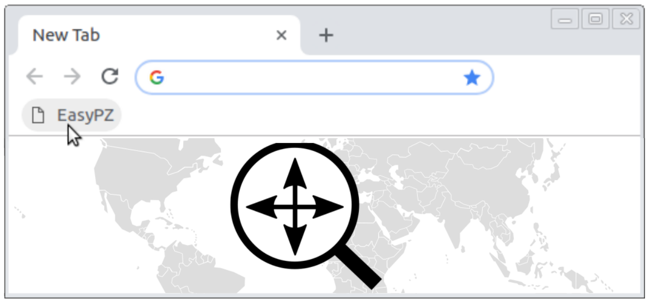


Figure 3: The EasyPZ Bookmarklet allows activating advanced pan and zoom techniques on visualizations on the Web with a simple click.

Here, EasyPZ provides the transformation and allows visualization creators to use this information to update their visualizations as desired. They still benefit from all of EasyPZ's features, including the flexibility to switch between methods and set their settings, customizability, and extendability. More information on the manual mode, including examples for updating D3.js scales and projections, and usage in canvas, can be found in documentation at `github.com/michaschwab/easypz/wiki`.

3.5 Implemented Interactions

With many methods implemented, visualization creators have more options to choose techniques compatible with other interactions, and have the opportunity to address more specific user needs, such as more advanced techniques being required for experienced users.

Pan: EasyPZ comes with standard desktop dragging with a linear transfer function (*drag pan*), standard mobile dragging with flick momentum (*flick pan* [17]), using the scroll wheel on a desktop mouse or a scroll gesture on a touchpad (*scroll pan*), and using the touchscreen with a two-finger motion (*two-finger pan*).

Zoom: EasyPZ comes with double click or tap with a primary or secondary button (*2x click zoom*), holding down a button or pressing a touchscreen without releasing the finger (*hold zoom* [4, 16]), using the scroll wheel on a desktop mouse or a scroll gesture on a touchpad (*scroll zoom*), two-finger pinch to zoom (*pinch zoom* [11]), drag-selecting a rectangular zoom region, (marquee zoom or *brush zoom*), dragging along the orthogonal axis for continuous zoom granularity control ('dynamic zoom' or *ortho zoom* [1, 3]), and the cyclic gesture of bi-directional rubbing (*rub zoom* [15]), e.g., where horizontal rubbing zooms in and vertical rubbing zooms out.

3.6 Bookmarklet

As a result of DOM-level interaction binding, where the interactions do not have to be tied to a visualization's JavaScript code, we can inject interactions into existing visualizations on the Web. Our EasyPZ browser bookmarklet, shown in Fig. 3 and demonstrated in our supplemental video, allows enabling pan and zoom on an existing SVG-based visualization during that session. The bookmarklet will inject the EasyPZ library into the website, detect any SVG-based visualizations, and enable EasyPZ with standard pan and zoom techniques. Users can even create customizable bookmarklets, through which users have control over technique selection and parameter tuning, e.g., to create designs for improved accessibility.

The bookmarklet can be compared with native browser zoom, which can also enlarge SVGs. Our approach has the following advantages: First, it shows zoom behavior as enabled through a JavaScript library, and so serves as an accurate preview of permanent zooming. Second, the zoom position cannot be specified with native browser zoom – every zoom operation requires re-orienting. Third, the EasyPZ bookmarklet preserves website content and font size and does not cause page layout disruptions. Fourth, the bookmarklet ensures that people do not get lost by limiting pan and zoom scope. These advantages are visible in the supplemental video.

4 Demonstration of Utility

To evaluate the coverage of our toolkit for existing visualizations, we sought a set of visualization examples from `bl.ocks.org`, mainly used by visualization developers (about 4 million unique visitors per year), and `reddit.com/r/dataisbeautiful`, mainly used by visualization viewers (13.3 million subscribers).

bl.ocks.org: We tested EasyPZ on the first 50 available examples on `bl.ocks.org`, sorting alphabetically. Of the 50, 39 worked with EasyPZ out of the box with the EasyPZ bookmarklet (see "Bookmarklet" in Fig. 1). Of the remaining 11, 8 did not work automatically because they used canvas or HTML instead of, or in combination with, SVG, but EasyPZ could work using a manual event handler with about 5 lines of code (see "Manual Scaling" in Fig. 1). Two visualizations had some interaction incompatibilities, such as already using zoom, where EasyPZ could be useful with a smaller set of additional techniques. The final visualization had a combination of preexisting transformations EasyPZ is not yet compatible with: rotated, translated bars. We demonstrate some of the examples at `demos.easypz.io`.

reddit.com/r/dataisbeautiful: We contacted the authors of 20 of the most viewed visualizations on the dataisbeautiful subreddit, 8 of which shared their source vector images with us. We demonstrate these more interactive beautiful data visualizations at `datais.easypz.io`. In all 8 cases, EasyPZ worked immediately.

Adoption: Since making EasyPZ.js available in January 2018, a first systematic evaluation of pan and zoom techniques has been published by the authors of this paper using EasyPZ.js [18]. As of July 2019, EasyPZ was downloaded 3,704 times from NPM and starred 67 times on GitHub.

5 Discussion and Limitations

EasyPZ automates the enabling of multi-scale interactivity as much as possible. However, developers still need to write manual implementations to apply transformations to advanced visualizations, such as to load data at different levels of granularity for performance. Still, with EasyPZ, authors do not need to implement the interactions or events, and are well equipped to change or extend the pan and zoom interactions they allow. This is helpful to allow mobile use, or to enable physically impaired persons.

Many existing technique implementations are not available, and many are poorly documented. While EasyPZ aids standardization for these techniques, the community must agree on a technique definition to confirm an implementation and its parameters, and use this definition to provide fair technique comparisons to inform visualization creators. While EasyPZ already supports many techniques, there are still techniques missing from EasyPZ. We hope that with documentation and the authoring tools we provide, interaction designers will use EasyPZ to contribute their interactions to an open-source platform.

Finally, even though we can add interactive control to static visualizations, rendering speed is still a limiting factor. A visualization too complex to draw at interactive rates will not be sped up by EasyPZ. We leave improving rendering performance for future work.

6 Conclusion

We have presented EasyPZ, an open-source JavaScript library that provides ways for the data visualization community to benefit from interaction improvements development in the human-interaction community. EasyPZ improves interaction development, standardization, tuning, evaluation, distribution, deployment, tuning, and adoption. By streamlining this process, data visualization users can leverage more advanced techniques that enable faster navigation, can choose interactions for more beginner or expert users or address physical limitations, and overall improve user experience.

Acknowledgments

The authors thank Alex Ahmed, Cody Dunne, Aditeya Pandey, and Yixuan Zhang for their helpful feedback on this paper. This work was supported in part by a Northeastern University Tier 1 Interdisciplinary Grant, and by the Khoury College of Computer Sciences at Northeastern University.

References

[1] C. Ahlberg and B. Shneiderman. The alphaslider: A compact and rapid selector. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pp. 365–371. ACM, New York, NY, USA, 1994. doi: 10.1145/191666.191790

[2] J. T. Alexander Schmitz, Chris Thoburn. Hammer.js.

[3] C. Appert and J.-D. Fekete. Orthozoom scroller: 1d multi-scale navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pp. 21–30. ACM, New York, NY, USA, 2006. doi: 10.1145/1124772.1124776

[4] B. B. Bederson and J. D. Hollan. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, UIST '94, pp. 17–26. ACM, New York, NY, USA, 1994. doi: 10.1145/192426.192435

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

[6] J. Choi, D. G. Park, Y. L. Wong, E. Fisher, and N. Elmqvist. Visdock: A toolkit for cross-cutting interactions in visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(9):1087–1100, Sept 2015. doi: 10.1109/TVCG.2015.2414454

[7] A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, Jan. 2009. doi: 10.1145/1456650.1456652

[8] G. W. Furnas and X. Zhang. Muse: A multiscale editor. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, pp. 107–116. ACM, New York, NY, USA, 1998. doi: 10.1145/288392.288579

[9] T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, pp. 139–148. ACM, New York, NY, USA, 2000. doi: 10.1145/354401.354435

[10] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister. Data-driven guides: Supporting expressive design for information graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):491–500, Jan. 2017. doi: 10.1109/TVCG.2016.2598620

[11] M. W. Krueger, T. Gionfriddo, and K. Hinrichsen. Videoplace&mdash;an artificial reality. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '85, pp. 35–40. ACM, New York, NY, USA, 1985. doi: 10.1145/317456.317463

[12] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 123:1–123:13. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3173697

[13] S. Malacria, E. Lecolinet, and Y. Guiard. Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: The cyclostar approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pp. 2615–2624. ACM, New York, NY, USA, 2010. doi: 10.1145/1753326.1753724

[14] T. Masui, K. Kashiwagi, and G. R. Borden, IV. Elastic graphical interfaces to precise data manipulation. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pp. 143–144. ACM, New York, NY, USA, 1995. doi: 10.1145/223355.223471

[15] A. Olwal, S. Feiner, and S. Heyman. Rubbing and tapping for precise and rapid selection on touch-screen displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pp. 295–304. ACM, New York, NY, USA, 2008. doi: 10.1145/1357054.1357105

[16] K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pp. 57–64. ACM, New York, NY, USA, 1993. doi: 10.1145/166117.166125

[17] P. Quinn, S. Malacria, and A. Cockburn. Touch scrolling transfer functions. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pp. 61–70. ACM, New York, NY, USA, 2013. doi: 10.1145/2501988.2501995

[18] M. Schwab, S. Hao, O. Vitek, J. Tompkin, J. Huang, and M. A. Borkin. Evaluating pan and zoom timelines and sliders. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 556:1–556:12. ACM, New York, NY, USA, 2019. doi: 10.1145/3290605.3300786

[19] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor. Dataink: Direct and creative data-oriented drawing. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, pp. D413:1–D413:1. ACM, New York, NY, USA, 2018. doi: 10.1145/3170427.3186471