Crossets: Manipulating Multiple Sliders by Crossing

Charles Perin.* INRIA / University of Calgary
Pierre Dragicevic.† INRIA
Jean-Daniel Fekete.‡ INRIA

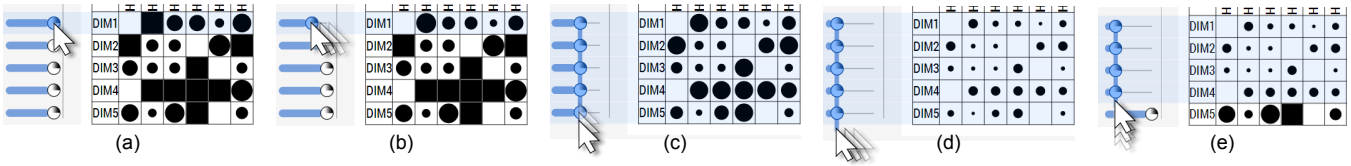

Figure 1: Crossing several continuous sliders (Crossets) in order to change the visual encoding of table cells: (a) selecting the thumb of a Crosset; (b) modifying the value associated to the selected Crosset by dragging the mouse left; (c) selecting a series of adjacent Crossets by dragging the mouse down; (d) modifying the value of all selected Crossets by dragging the mouse left; (e) deselecting a Crosset by dragging the mouse up, restoring its associated value to the one before selection.

Abstract

Crossets are new interactive instruments or widgets based on crossing gestures that exploit the dimension orthogonal to sliders' axes for manipulating multiple aligned sliders simultaneously. We propose a Crossets taxonomy to generalize the sliders' properties to those of other standard widgets. We introduce and illustrate the constrained crossing gesture with Crossets in an interface for the visual exploration of numerical tables. Then, we discuss alternative strategies to Crossets before exploring persistent unconstrained crossing gestures compatible with Crossets, introducing Spline as a persistent reusable interactive instrument. This paper highlights promising perspectives for crossing-based widgets. We hope future interfaces will make use of this simple technique that can help improve the efficiency of standard widgets and lead to the generation of new styles of interfaces.

Index Terms: H.5.2 [User Interfaces]: Graphical user interfaces (GUI)—User interfaces

1 Introduction

This paper introduces Crossets—crossable widgets—for manipulating multiple sliders simultaneously, from tens to hundreds. The design rationale for Crossets is based on principles of direct manipulation [26, 39] and on the instrumental interaction model [8]. The basic Crosset is a slider enhanced with crossing capabilities, that generalizes to other widgets (an early version of this article has been published in French [33]).

Most standard GUIs use only one dimension of the two-dimensional space of the screen and one dimension of the mouse to interact with sliders, while some other works use the orthogonal dimension for the sliders' axis [4, 16]. Despite the fact that crossing [1] for selecting several objects in a single gesture is well suited to the simultaneous manipulation of multiple sliders, this technique has never been applied to manipulate the orthogonal dimension of sliders (illustrated in Figure 1). While pointing requires to target an object and press the mouse button to activate it, crossing consists of pressing the mouse button and moving the cursor over a series of objects to activate them all.

Crossets are beneficial for interfaces presenting objects that are *similar, aligned, and on which users often perform the same small set of actions over a range of consecutive objects*. Two main categories of applications can benefit from using Crossets. First, *list-based* applications such as email readers and file explorers. For example, managing emails often involves selecting multiple messages and invoking some command (e.g., mark as spam or as important). Crossets would be a great complement to these tools. As another example, recent files in a folder could be labelled by first sorting them by date, and then rapidly switching their color with a Crosset. Second, *table-based* applications such as spreadsheet calculators would benefit from Crossets for applying specific formatting/coloring to a subset of rows (or columns). Currently, users must select the rows on which to apply the action, then open a menu and apply the action using a selection-action*-deselection cycle. Crossets merge the selection-action-deselection in a single interaction, and make it possible to apply the action directly with continuous feedback.

In this article, we first introduce a taxonomy of Crossets based on standard widgets. We present the *constrained* crossing gesture, orthogonal to the sliders' axis, and we illustrate the benefits of Crossets with a complex visualization application relying heavily on Crossets [34]. In this application, the gesture interpretation by the system is constrained to be orthogonal to the widgets' axes and compatible with the standard mouse input device. This example illustrates both benefits and limits of this constrained crossing gesture. Then, we propose and discuss advanced crossing gestures that are compatible with Crossets, opening the discussion towards a new persistent, reusable, and more expressive interactive instrument than the constrained gesture called the *Spline*. In summary, the contributions of this paper are:

- The implementation of a constrained crossing gesture compatible with, and enhancing, standard widgets.
- A taxonomy of Crossets based on the slider.
- The illustration of the constrained crossing gesture with an implemented tabular visualization system.
- The exploration of persistent unconstrained crossing gestures and the introduction of the *Spline* instrument, opening promising perspectives for future research and new interfaces.

*e-mail: charles.perin@ucalgary.ca.
†e-mail: pierre.dragicevic@inria.fr.
‡e-mail:jean-daniel.fekete@inria.fr.

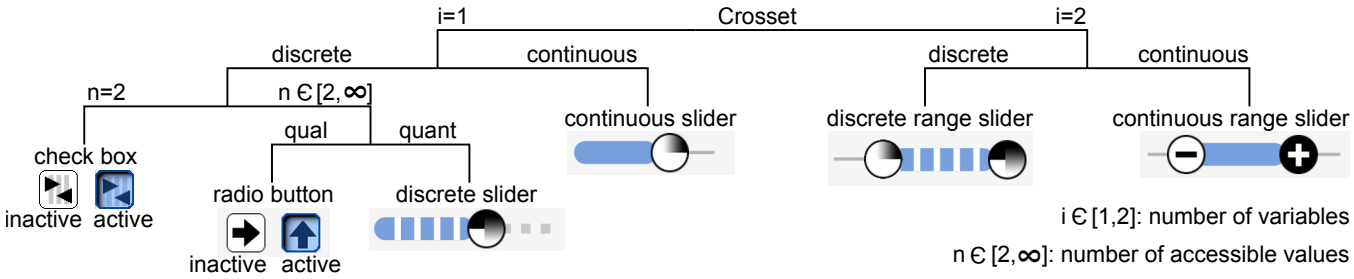

Figure 2: Taxonomy of Crossets and implementation, based on the slider.

2 Background

Sliders and scrollbars are used in many interfaces, e. g., to navigate time in video players, to navigate into viewports, and to set values from valid ranges, such as setting color values in photo editors.

2.1 Two-dimensional Sliders

The instrumental interaction framework [8] can help explain how the design of sliders can be improved by introducing the *degree of integration*. This measure is the ratio between the number of degrees of freedom used to manipulate the instrument and the number of degrees of freedom of the physical input device (e. g., the mouse).

While instruments should ideally have a degree of integration of 1, sliders have a degree of integration of 1/2. Thus far, the two spatial dimensions have been used to interact with sliders: the Infovis toolkit [16] features multi-resolution sliders with precision dependent on the orthogonal distance to the slider. Orthozoom [4] extends the scrollbar by controlling both pan, along the scrollbar's axis, and zoom, along the orthogonal axis. FaST Sliders [31] exploit the orthogonal dimension of sliders to make an additional menu pop up for further adjustments. As of yet, the orthogonal dimension has not been used to manipulate multiple sliders.

2.2 Crossing

Crossing-based interfaces let users invoke commands by crossing (or painting over) widgets instead of clicking them [3]. Crossing is used to drag and drop icons in background windows [13] and to successively select elements in hierarchical menus [2]. Crossing has also been found efficient as a standard point-and-click interaction for selecting multiple targets in dialog boxes [12]. Similarly, Baudisch proposes crossing-based interactions for activating check-boxes [7], which were later generalized to a set of standard widgets with Sliding Widgets [32], for manipulating *individual* widgets by crossing. Although Sliding Widgets make it possible to manipulate two sliders using a pointing area, the orthogonal dimension is not exploited to manipulate more than two widgets at a time.

Crossable widgets already exist. For example, Photoshop lets users apply the same action to several layers by pressing an icon and dragging the mouse cursor to other layers. However, the gesture is not reversible and requires pointing at as many targets as there are icons. Moreover, the gesture is available for binary values (layer visibility), but has not been extended to continuous values (e. g., layer opacity).

Finally, the most efficient crossing gestures are continuous and performed orthogonally to the crossed object [1, 3]. However, crossing requires performing a steering task (e. g., to cross multiple check-boxes), and steering in a narrow tunnel (dragging along a precise path) is a slow motor task [1, 3].

To summarize, the slider is a ubiquitous widget but it suffers from the following limitations:

- It uses only one dimension of two-dimensional space,
- Setting the same value to several sliders is tedious,
- The graphical alignment of multiple sliders is not exploited.

3 Constrained Crossing Gesture

The basic Crosset is an extension of the slider widget, enhanced with orthogonal crossing capabilities. Figure 1 illustrates the technique with continuous sliders: (a) selecting a slider's thumb activates the gesture; (b) moving the mouse cursor horizontally sets the Crosset value (here the visual encoding of the cells is updated); (c) moving the mouse cursor vertically copies the value to other Crossets and adds them to the list of Crossets to be controlled; (d) moving the mouse horizontally changes the value of all crossed widgets; and (e) uncrossing a widget restores its value before selection.

Both dimensions of the plane correspond to one of the following domains:

Value domain: The domain mapped to the Crosset axis. For example, the value domain of a slider is its range of possible values, and the value domain of a group of radio buttons is the discrete set of buttons. To be compatible, adjacent Crossets must have the same value domain.

Selection domain: The domain mapped to the dimension orthogonal to the Crosset axis. To make a selection among several Crossets, the Crossets must have the same type (e. g., sliders only).

The constrained crossing gesture makes it possible to navigate in both the value domain and the selection domain and is dedicated to the simultaneous manipulation of adjacent sliders in a continuous manner. Despite the fact that the gesture is freely drawn using e. g., a mouse cursor, its interpretation by the system is constrained to the axis orthogonal to the sliders' axes to ensure a fast and errorless gesture. It is also possible to adjust both the selection and the values continuously, therefore eliminating traditional selection–action sequential interactions. This implementation state automaton has three states:

Start: The crossing is enabled when pressing the mouse button on the thumb/button of a Crosset. The thumb/button visual appearance is updated to confirm that it is selected.

Drag: Moving the mouse in two dimensions lets users navigate both in the value domain and in the Crossets selection domain. The selected Crossets are all highlighted, and an orthogonal line linking the first selected Crosset to the mouse cursor shows how crossing has been interpreted by the system. The same value is applied to the selected Crossets, and unselecting a Crosset restores its value before selection.

End: Manipulation terminates when releasing the mouse button.

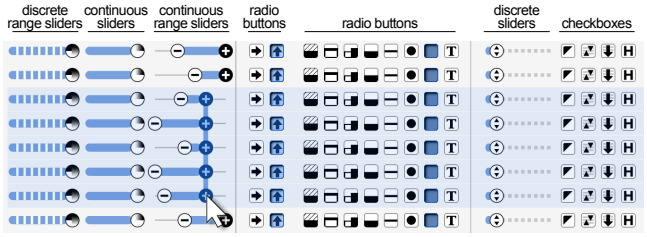

Figure 3: Example of an interface based on Crossets. The axes of all Crossets are horizontal, and crossing interpretations are vertical. Here, several continuous range sliders have their maximum value simultaneously set.

3.1 Taxonomy of Crossets

While the basic Crosset is an enhanced slider, a range of widgets can become new Crossets as well. Figure 2 illustrates the taxonomy of Crossets that we have implemented. Let i be the number of variables controlled by the Crosset. In most cases, $i \in [1,2]$ (e.g. a regular slider or a range-slider), but sometimes $i > 2$. For example, Photoshop features 3-thumbs sliders to define color gradients. Let $n \in [2,\infty]$ be the number of distinct values reachable by the Crosset (size of the domain of values). We implemented the following Crossets:

- The basic Crosset is a continuous slider ($i = 1, n \gg 1$).
- The discrete slider is a continuous slider snapping to discrete positions ($i = 1, n > 1$).
- The continuous range slider ($i = 2, n \gg 1$) is a continuous slider with two thumbs controlling a min and a max value.
- The discrete range slider is a continuous range slider ($i = 2, n > 1$) snapping to discrete positions. It inherits the properties of both the discrete slider and the range slider.
- A group of radio buttons is similar to the discrete slider ($i = 1, n > 1$) but is dedicated to non-ordinal (nominal) values. Each button can be seen as a step of a discrete slider and the selection consists of one of these values.
- The check-box ($i = 1, n = 2$) has two states. It is similar to existing implementations [7, 32].

Figure 3 shows an interface where Crossets are grouped and stacked vertically. The orthogonal dimension of Crossets that do not have a clear axis (e. g., check-boxes) is the one on which the Crossets selection is done. Here, all Crossets' axes and domain of values are horizontal.

3.2 Properties of Crossets

Crossets align with several principles issued from direct manipulation [26, 39] and instrumental interaction [8]:

Internal Consistency: A wide range of standard widgets can be converted to Crossets. This ensures the internal consistency of the interface—the consistency of a design with itself [21], in terms of interaction, design and graphical properties. Internal consistency favors initial learning, ease of use, and perceived quality [21].

External Consistency: Measures the consistence of an interface regarding other existing interfaces [21]. Crossets have a high external consistency because they are compatible with standard interactions, only providing enhancements. In particular, Crossets are compatible with the click state automaton, consisting of the **Start** state followed immediately by the **End** ignoring the **Drag**s. The crossing gesture is triggered only when the user takes a trajectory orthogonally to the widget's axis. External consistency favors fast learning and transfer of knowledge between UIs [21, 26, 39]. Moreover, crossing-based interfaces are easier to accept if they are compatible with the standard point-and-click paradigm [12].

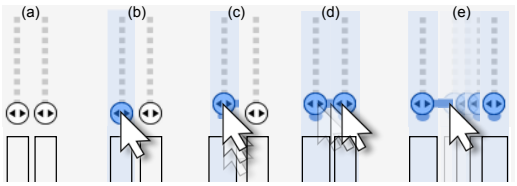

Figure 4: Issues occur when modifying the value of a Crosset affects the position of one or several Crossets.

Degree of Integration: In contrast with standard widgets which have a spatial degree of integration of 0/2 (e. g., buttons) or 1/2 (e. g., sliders), Crossets have a degree of integration of 2/2 because the two spatial dimensions of the mouse are exploited.

Incremental and Reversible Actions: The gesture allows for incremental updates and is reversible, one of the basic principles of direct manipulation interfaces [39], as illustrated in Figure 1(d,e).

Support for Multiple Objects: By navigating in the Crosset's selection domain, the gesture lets users apply the same action to a series of objects, one of the challenges for direct manipulation interfaces [18, 19, 20].

Simple Gesture: As crossing is performed in only one dimension of the two-dimensional space, the gesture is relatively free but its interpretation by the system is constrained in the direction orthogonal to the Crosset axis. This property avoids steering tasks over narrow targets and ensures no selection errors [1], without the need for intrusive error messages that can interfere with the task at hand [26, 38].

3.3 A Crosset-based Interface

To illustrate the technique (constrained crossing gesture) and discuss practical design considerations and issues, we refer in this this section to Bertifier, an interface dedicated to the visual exploration of numerical tables [34] available at `www.bertifier.com`.

This section is not meant to be a formal evaluation of Crossets, but rather a conceptual discussion. Moreover, the evaluation of Bertifier involves the integration of Crossets within the whole interface and visualization.

Bertifier is based on Crossets, however, the Crossets technique, taxonomy, and rationale are not detailed in the related previous publication [34]. Bertifier is a tabular visualization in which Crossets are placed next to rows and columns, making it possible to perform actions on arbitrary groups of adjacent rows and columns in a quick and well-integrated manner. Thus, this interface is a good example for illustrating *table-based* interfaces, the first category of interfaces that can benefit from Crossets.

Large Number of Crossets: UI Guidelines recommend avoiding too many widgets [42], however, the multiplication of Crossets is a strength of Bertifier. Indeed, it addresses the challenge of applying an action on several objects of interest by one single interaction [19]. Moreover, applying actions does not require specification

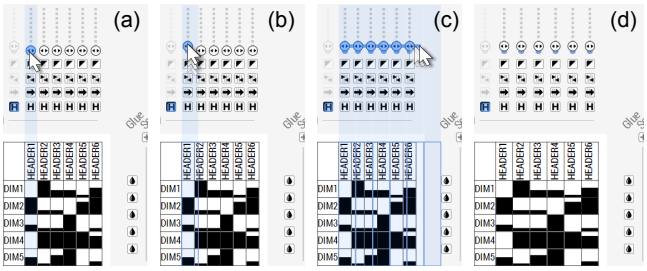

Figure 5: Preview as immediate feedback for column resizing.

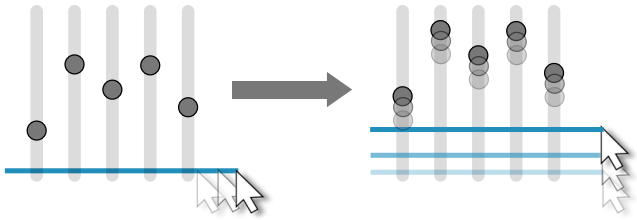

Figure 6: Relative dragging to maintain the sliders' values offset.

of the selection in advance—it is made on-the-fly—as opposed to the traditional selection-action sequential approach. Additionally, the gesture's reversibility makes the system tolerant to errors. Also, as objects of interest are rows and columns, duplicating Crossets makes it possible to align Crossets with their associated object of interest.

Low Spatial Indirectness of Crossets: As Crossets are aligned with their target row or column, their spatial offset is null according to one of the two dimensions of the space. Thus, their spatial indirectness [8] is lower than widgets not designed to be aligned with the objects of interest and located far from them.

Low Temporal Indirectness: In Bertifier, the selection of objects of interest and the manipulation of values are performed with a unique gesture with immediate feedback [39, 26], and the actions are instantaneously applied to the objects of interest. Thus, Crossets minimize temporal indirectness [8] and articulatory distance [26], with the exception of Crossets affecting the position of other Crossets, as we explain below.

Degree of Compatibility: Designing an entire interface involves tradeoffs, and internal consistency can conflict with the degree of compatibility [8] of some Crossets. For example, to modify the width of a column, the user must drag the slider thumb up and down, while horizontal dragging would be more cognitively congruent (i. e. similarity between the intent of the user and the action she performs).

Scalability: A recognized challenge of direct manipulation interfaces is targeting invisible objects [19]. Crossets must be visible in the viewport to be reachable, which may be a problem when the table is too large, requiring users to perform several gestures coupled with page scrolling. However, zooming out solves the problem in most cases.

Layout Modifications: When the action associated to a Crosset affects the position of one or more Crossets (e.g. changing the size of the columns of a spreadsheet), it may be too disruptive to apply the result of this action on-the-fly. Indeed, let us consider the scenario illustrated in Figure 4. Here, each Crosset can be used to resize a column (shown as black rectangles): dragging up the Crosset's thumb enlarges the column. Modifying the value of the first Crosset is not problematic (b,c). However, crossing the second Crosset (c_2) increases the width of the second column, making c_2 move away from the mouse cursor and c_2 becomes deselected (d,e). The value of c_2 is then reset, c_2 moves back to its initial position, under the mouse pointer, and is selected again. As a result, c_2 and its associated column are undesirably oscillating from left to right.

Thus, in Bertifier, for Crossets' actions affecting the position of other Crossets, a partial feedback is provided during the **Drag** phase (Figure 5(a,b,c)) and the final result is only applied once the mouse button is released, corresponding to the crossing gesture **End** (Figure 5(d)). It ensures compatibility with actions affecting layout, at the cost of reducing temporal directness [8]. The strategy is generic and applied to all actions affecting the table layout.

User Feedback: Results of a user study involving using Bertifier emphasized the benefits of Crossets that participants found both effective and efficient [34]. Our previous article emphasizes the fact that no participant had difficulties understanding and performing the cross-widget gesture, thanks to the external consistency of Crossets. Participants did not complain about the low degree of compatibility of some Crossets (e. g., row and column resizing). Overall, they found the interface playful and easy to use, thanks to its high internal consistency. Bertifier is regularly used today in the context of academic courses for data exploration, and also by many researchers from various research fields (e. g., humanities and social sciences) who need to visualize, analyze, and communicate their data. We believe that Crossets are one important reason why Bertifier successfully revived a previous method for interactively exploring tabular data while many other attempts, rooted in the WIMP model and featuring selections and series of menus and buttons, failed previously.

3.4 Limitations

Designing Bertifier, an interface based only on Crossets, highlighted several limitations of Crossets:

Layout stability: To ensure interface layout stability when applying layout actions, we chose to provide immediate but partial feedback as a preview of the result, and apply the modifications at the end of the interaction. With this delayed activation, Crossets do not affect the layout stability, at the cost of some temporal indirectness.

Scalability: Crossets must be visible to be selected, which raises the issue of accessing objects outside of the viewport. In the case of Bertifier, we solved this problem by zooming out. Note that most spreadsheet programs support zooming as well, but often, large tables cannot be zoomed out to fit all their rows in the viewport. Crossets could be improved by implementing auto-scrolling in these cases.

Skipping Crossets: As the crossing gesture is orthogonal to the Crosset axis, there is no way of avoiding a Crosset on the pointer's path. Adding a key modifier to "skip" a Crosset may solve this issue at the cost of adding complexity to the interaction, when simplicity is one of its strengths. A possible solution consists of memorizing the last slider value set, or the last selection range(s). When resuming a crossing gesture with a key modifier, the values selected would either snap to the memorized value, or extend the previous selection to allow direct manipulation of non-adjacent Crossets. Both cases require a key modifier.

Relative dragging: With the constrained crossing gesture, all selected Crossets are assigned the same value. One interesting alternative consists of invoking a relative dragging, in which selected Crossets maintain their relative offset. This alternative is illustrated in Figure 6: selecting sliders does not change their values, but instead navigating in the value domain (vertically in the Figure) maintains the relative offset between sliders' values. The drawback of this alternative is that it also requires activating a new mode, reducing the simplicity of the interaction.

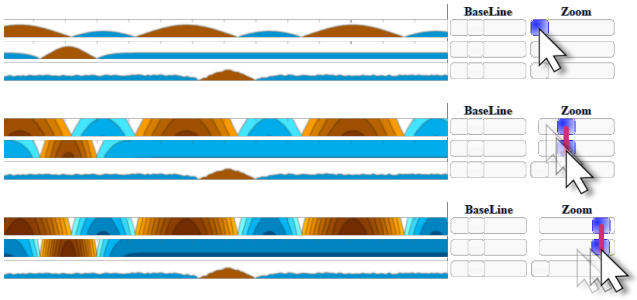

Figure 7: Setting the zoom factor of two Horizon Graphs using Crossets. From top to bottom, the zoom factor is increased, resulting in increasing numbers of bands.

3.5 Applications

We do not claim that Crosset-based operations would be useful in all applications, however we postulate that in appropriate cases, they can be effective. Crossets can enrich many user interfaces at no extra cost in usability or screen real estate when widgets are already used. We discuss two large categories of interfaces that would benefit from Crossets: *table-based* and *list-based* interfaces.

Table-Based Interfaces: Bertifier belongs to the category of *table-based* interfaces. Table-based applications include software such as spreadsheet editors and interactive tabular visualizations (e. g., TableLens [37] and InfoZoom [40]), as well as calendar views, adjacency matrices (e. g., MatrixExplorer [23] and Cubix [6]) and scatterplot matrices (e. g., ScatterDice [14]). Such applications often require applying identical actions to series of rows or columns.

For example, Crossets could be used to expand/collapse (or change the weight of) arbitrary sets of rows and columns in an adjacency matrix. In table-based applications, Crossets can be laid out around the table, both near to rows, where the selection axis is vertical, and near to columns, where the selection axis is horizontal, such as in Bertifier.

List-Based Interfaces: The second category of interfaces that would benefit from Crossets is *list-based*, i. e. interfaces that employ a list layout. In such interfaces, Crossets can be laid out near each entry of the list. These include file explorers, email readers, layer-based authoring applications, and stacked visualizations.

Considering the example of layers of authoring software, the same effect often needs to be applied to several video or audio tracks. Photoshop already has crossable toggle buttons to hide or unhide layers. Users are free to rearrange layers or tracks semantically in order to optimize their manipulation with Crossets.

Another example we implemented is illustrated in Figure 7. Interactive Horizon Graphs [35] is a technique for visualizing multiple stacked time series. It extends and improves Horizon Graphs [17, 22] by making it possible to interact with two parameters: i) the baseline panning consists of setting the value of a baseline separating values above and below it (values in blue and brown in Figure 7, respectively); ii) the zoom factor consists of wrapping the chart around its frame, creating additional bands, a value being estimated by both its height and its hue/saturation. In this example, Crossets make it possible to set the same baseline value or zoom factor to several time series. As in Bertifier, dragging and dropping time series to change their layout is necessary to spatially group similar time series of interest to be modified simultaneously. Moreover, in this touch-compatible version, both parameters can be manipulated at the same time to finely adjust them. Stacked time series visualizations may also directly benefit from other Crossets, such as crossable radio buttons to select the type of representation of the time series (e. g., inverting the baseline or not) and the colors to be used, crossable range

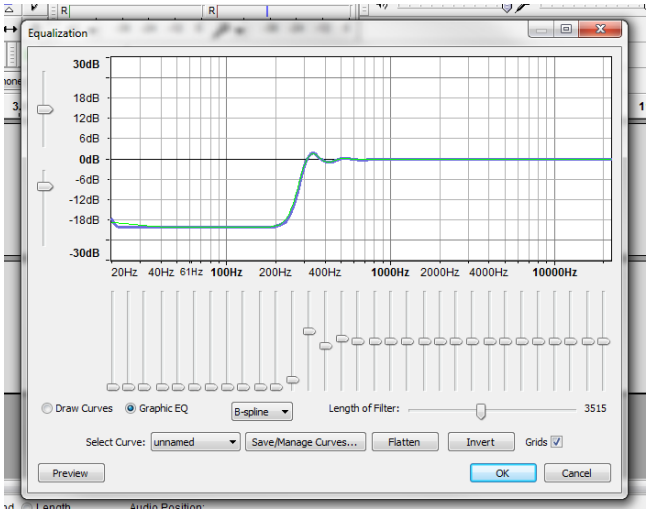

Figure 8: The equalizer in Audacity. Two views are available: curves (top) and sliders (bottom). Here, a standard bass-cut filter is applied.

sliders to adjust the temporal zoom (i. e. specify a minimum and a maximum time to be represented), and sort operations to reorder a subset of time series.

3.6 Noun-verb vs Verb-noun

Traditionally, interfaces provide a noun-verb sequence: objects are selected first, then an action is applied to the selected objects. The inverse is the verb-noun sequence, where the action is specified first, and then a selection is performed. For example, in a vector graphics editor, selecting several shapes then specifying their stroke width is a noun-verb sequence. Inversely, selecting a style then applying this style to a set of shapes is a verb-noun sequence.

Crossets do not require such a choice. Indeed, both the selection (noun) and the values (verb) are set simultaneously in a continuous gesture, accelerating the acquisition of expert operational skills [11]. As a result, Crossets are particularly effective when the nouns are sets of objects that often change over time, requiring the user to perform multiple selections. Conversely, if actions need to be repeatedly applied to the same groups of objects, the constrained crossing gesture we propose is not the best option. Instead, maintaining selections using inspector windows or surrogate objects [25, 29] can be a better alternative to apply successive actions to the same selection. Groups are widely used and extremely powerful, but it often occurs that they must be un-grouped or re-grouped in order to carry out an action. We believe these two approaches are compatible and complementary.

4 Persistent Unconstrained Crossing Gesture

By using the orthogonal dimension of the Crosset axis, with the selection axis being independent of the manipulation axis [27], Crossets add fluidity to interaction for frequent tasks [15, 28]. The constrained crossing gesture is one of the possibilities that Crossets offer, with the benefit of unifying manipulation and selection in a single gesture [11]. The gesture is interpreted by the system orthogonally to the Crosset axis, making it possible to select and adjust sliders with a loose gesture and to refine the values afterwards. In particular, this strategy is efficient when Crossets are of discrete type (e. g., radio-buttons and check-boxes).

However, the caveat is that this simple gesture introduces a constraint in the value domain, requiring all the selected widgets to be set to the same value, thus reducing the expressiveness of the

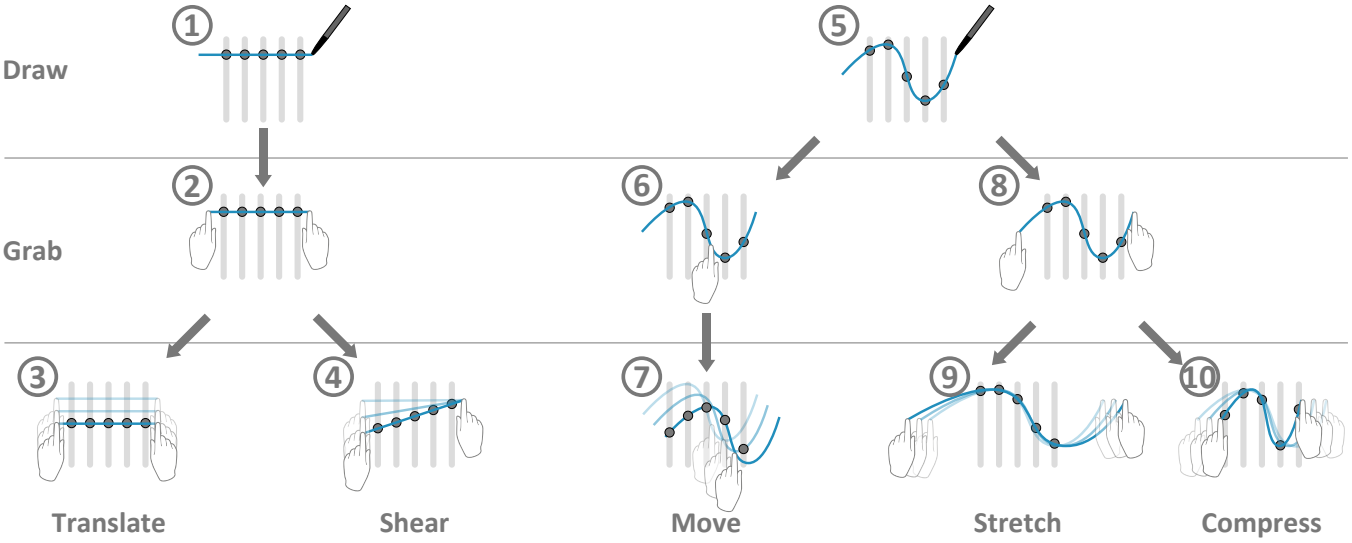

Figure 9: Possibilities of pen and touch interactions to manipulate persistent selections.

interaction. We considered several alternatives with their tradeoffs for manipulating Crossets:

1. The *constrained gesture* assigns the same value to all the selected Crossets. The gesture is simple and errorless, and allows for a loose gesture to rapidly assign the same value to several objects of interest at the cost of lower expressiveness.

2. The *unconstrained gesture* lets one freely draw a trajectory in two-dimensional space, selecting the starting value of each slider at the last crossing point. This is the standard approach of crossing interactions. This gesture offers more expressiveness, but requires precisely setting each selected slider's value. Moreover, adjusting a slider's value afterwards requires redrawing the trajectory entirely.

3. The *persistent unconstrained gesture* transforms the crossing gesture into a reusable, editable object (reifying the selection). This alternative is as expressive as the unconstrained gesture, but maintains the selection for further editing and reuse. It also opens a new world of possible interactions on the reified selection.

Reifying selections (e. g., copy-paste a selection object to apply it to another set of Crossets) can leverage Crossets to make them more expressive and powerful [9]. Once Crosset selections are considered as first-class objects, tools like lassos and magnetic guidelines [36, 41] could be used on them. These tools make the selection persistent and thus reusable, but they are not always compatible with crossing gestures and they are not easily reversible. Next, we detail the persistent unconstrained gesture: the *Spline* instrument. The Spline instrument provides interactions that are expressive, reusable, and editable, while avoiding the tradeoff of noun-verb vs. verb-noun approaches—it allows both.

4.1 The Spline Instrument

Similarly to Toolglass and Magic Lenses [10], the two-dimensional unconstrained drawing itself can become a persistent interactive instrument, where the shape of the selection can be manipulated, modified, and reused. Basically, after drawing a two-dimensional Spline both in the selection domain and in the value domain (an unconstrained crossing gesture), the idea consists of making this two-dimensional Spline persistent so that the selection and Crossets' associated values do not disappear. The novelty here is not the unconstrained crossing gesture, but the persistence of the selection. Such persistent selection makes it possible to reproduce and modify patterns representing a strong coupling between selection and values, with the following properties:

Unconstrained gesture: To assign in a single gesture different values to different Crossets by drawing a trajectory in two dimensions, similar to CrossY [2], in an unconstrained two-dimensional crossing gesture.

Selection recall: To embed the benefits of both Crossets (continuous navigation in both the value domain and the selection domain) while tackling the challenge of selection recall [24].

Continuous action: To specify both the noun and verb simultaneously and iteratively, while remaining reversible.

Expressive instrument: Making use of input modalities, such as pen and touch, that offer more powerful, diverse, and expressive gestures than the mouse to leverage Crossets, as these input technologies are compliant with crossing [30].

4.2 Proof of Concept

To illustrate the potential of the Spline instrument, consider the manipulation of an audio equalizer. An equalizer is a device used to achieve equalization, which consists of *"altering the frequency responses of an audio system using linear filters"* [43], in recording studios and during concerts. A basic equalizer consists of several aligned frequency-specific volume knobs to adjust the signals at particular frequencies. This example is particularly appropriate because frequencies are usually not assigned precise values, but are instead adjusted continuously using the resulting sound as feedback. Moreover, sliders are manipulated in groups of frequencies. Figure 8 shows the equalizer available in the open-source software Audacity [5]. This standard equalizer provides two alternative views: the curve for drawing a spline, and the corresponding sliders, with low frequencies on the left and high frequencies on the right, the vertical axis mapping the volume gain. While these two views are not superimposed, this interface could benefit from Crossets in order to merge the two views, and also make it more expressive.

To illustrate how Crossets and the Spline instrument can benefit equalizers, we explain how common tasks and operations are made

easier using this technique than a standard interface such as the one illustrated in Figure 8. A prototype of the Spline-based equalizer is available at `www.aviz.fr/intrinseq`.

The first standard operation consists of reducing or increasing the volume of all frequencies simultaneously, e. g., to reduce saturation. Instead of manually setting a smaller value to one slider at a time, the constrained crossing gesture is more efficient if all frequencies have the same original value (① in Figure 9). As the Spline instrument enhances the constrained crossing gesture, it also makes this action possible (②,③ in Figure 9). However, if frequencies have different values, relative dragging is required (⑤ in Figure 9). With the persistent Spline instrument, this simply consists of translating the selection up and down (⑥,⑦ in Figure 9).

Another common operation consists of creating a band-pass filter, consisting of two cutoff values and a bandwidth. Once a band-pass filter has been created, it usually needs to be adjusted. Instead of redrawing the Spline completely or setting each slider individually, the Spline instrument makes it possible to perform several meaningful and useful actions: stretch and compress the Spline (⑧,⑨,⑩ in Figure 9) to increase or decrease the bandwidth (stretching the curved line to its maximum transforms it into a straight line); move the Spline vertically (⑦ in Figure 9) to change both cutoffs and emphasize a new range of frequencies, making it possible to simply produce audio effects, such as the ones produced by a wah-wah pedal.

To illustrate other application examples, the shear gesture (④ in Figure 9) would be useful to apply an opacity pattern to several successive layers in Photoshop, e. g., to rapidly create a motion effect similar to the cursors and hands used in the Figures in this paper. Finally, all of the illustrated gestures in Figure 9 would provide interesting ways of navigating into the RGB domain to select colors with similar dependencies and create consistent color schemes.

5 Conclusion

We presented Crossets, new widgets for the simultaneous modification of multiple widgets similar to sliders. Crossets enhance standard widgets with crossing capabilities while avoiding steering tasks on narrow paths, which can be a slow motor task. Instead, the selection of targets is made easier by constraining the gesture interpretation to the orthogonal dimension of the widgets. This constrained crossing gesture makes it possible to specify in a continuous manner (single gesture) both the noun (the selection of object to be manipulated) and the verb (the action to apply on the selection).

The benefits of this constrained crossing gesture lie in its improved tradeoff between simplicity and expressive power compared to traditional slider-based interfaces. Further increasing this power—while possible, e. g., using relative dragging or making it possible to skip Crossets—also increases the complexity of the interaction. Facing this tradeoff between limited but simple, fast, and errorless gestures, and powerful and complex gestures, we introduced the *Spline* as a new persistent interactive instrument. The Spline instrument maintains the selection and is both reifiable and reusable, and makes possible the continuous and simultaneous interaction in both the selection domain and the value domain in a well-integrated manner.

Crossets are particularly applicable to table-based and list-based interfaces. These can greatly benefit from Crossets since they *present multiple similar objects that are graphically aligned, and on which users often perform the same small set of actions over a range of consecutive objects*. Examples include spreadsheet or visualization applications, virtual mixing consoles, and more general interfaces such as file explorers and email browsers.

Crossable widgets open new perspectives, and we hope more interfaces will take advantage of them in the future. In particular, transferring widget-based interfaces to Crosset-based interfaces would require little adaptation time from users, as Crossets interactions are compatible with standard widgets. Crossets also have a generative power, as they make it possible to completely remodel some aspects of WIMP interfaces that are tedious to use, and generate new interfaces for managing multiple objects based on a consistent interaction paradigm.

Acknowledgements

We are grateful to R3, who reviewed an early version of this work submitted to CHI 2015 and judiciously pointed out the flaws of the paper, thus helping us to greatly improve the quality of this work. We also thank Lindsay MacDonald for her thorough proofread of the camera-ready paper.

References

[1] J. Accot and S. Zhai. More than dotting the i's — foundations for crossing-based interfaces. In *Proc. CHI '02*, pages 73–80. ACM, 2002.

[2] G. Apitz and F. Guimbretière. Crossy: A crossing-based drawing application. In *Proc. UIST '04*, pages 3–12. ACM, 2004.

[3] G. Apitz, F. Guimbretière, and S. Zhai. Foundations for designing and evaluating user interfaces based on the crossing paradigm. *ACM Trans. Comput.-Hum. Interact.*, 17(2):9:1–9:42, May 2008.

[4] C. Appert and J.-D. Fekete. Orthozoom scroller: 1d multi-scale navigation. In *Proc. CHI '06*, pages 21–30. ACM, 2006.

[5] Audacity, 2014. Available at `http://audacity.sourceforge.net/`.

[6] B. Bach, E. Pietriga, and J.-D. Fekete. Visualizing dynamic networks with matrix cubes. In *Proc. CHI '14*, pages 877–886. ACM, 2014.

[7] P. Baudisch. Don't click, paint! using toggle maps to manipulate sets of toggle switches. In *Proc. UIST '98*, pages 65–66. ACM, 1998.

[8] M. Beaudouin-Lafon. Instrumental interaction: An interaction model for designing post-wimp user interfaces. In *Proc. CHI '00*, pages 446–453. ACM, 2000.

[9] M. Beaudouin-Lafon and W. E. Mackay. Reification, polymorphism and reuse: Three principles for designing visual interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 102–109. ACM, 2000.

[10] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 73–80. ACM, 1993.

[11] W. A. S. Buxton. Chunking and phrasing and the design of human-computer dialogues. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, editors, *Human-computer Interaction*, pages 494–499. Morgan Kaufmann Publishers Inc., 1995.

[12] M. Dixon, F. Guimbretière, and N. Chen. Optimal parameters for efficient crossing-based dialog boxes. In *Proc. CHI '08*, pages 1623–1632. ACM, 2008.

[13] P. Dragicevic. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proc. UIST '04*, pages 193–196. ACM, 2004.

[14] N. Elmqvist, P. Dragicevic, and J. Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE TVCG*, 14(6):1539–1148, Nov 2008.

[15] N. Elmqvist, A. Moere, H. Jetter, D. Cernea, H. Reiterer, and T. Jankun-Kelly. Fluid interaction for information visualization. *Information Visualization*, 10(4):327–340, 2011.

[16] J.-D. Fekete. The infovis toolkit. In *Proc. INFOVIS '04*, pages 167–174. IEEE, 2004.

[17] S. Few. Time on the horizon. available online at `http://www.perceptualedge.com/articles/visual_business_intelligence/time_on_the_horizon.pdf`, 2008.

[18] D. M. Frohlich. The history and future of direct manipulation. *Behaviour & Information Technology*, 12(6):315–329, 1993.

[19] D. M. Frohlich. Direct manipulation and other lessons. In *Handbook of human-computer interaction (2nd ed)*, pages 463–488. Elsevier, 1997.

[20] D. Gentner and J. Nielsen. The anti-mac interface. *Commun. ACM*, 39(8):70–82, Aug. 1996.

[21] J. Grudin. The case against user interface consistency. *Commun. ACM*, 32(10):1164–1173, Oct. 1989.

[22] J. Heer, N. Kong, and M. Agrawala. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *Proc. CHI '09*, pages 1303–1312. ACM, 2009.

[23] N. Henry and J. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE TVCG*, 12(5):677–684, Sept 2006.

[24] K. Hinckley, F. Guimbretiere, M. Agrawala, G. Apitz, and N. Chen. Phrasing techniques for multi-stroke selection gestures. In *Proc. GI '06*, pages 147–154. Canadian Information Processing Society, 2006.

[25] R. Hoarau and S. Conversy. Augmenting the scope of interactions with implicit and explicit graphical structures. In *Proc. CHI '12*, pages 1937–1946. ACM, 2012.

[26] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Hum.-Comput. Interact.*, 1(4):311–338, 1985.

[27] R. J. K. Jacob, L. E. Sibert, D. C. McFarlane, and M. P. Mullen, Jr. Integrality and separability of input devices. *ACM Trans. Comput.-Hum. Interact.*, 1(1):3–26, 1994.

[28] G. Kurtenbach and W. Buxton. Issues in combining marking and direct manipulation techniques. In *Proc. UIST '91*, pages 137–144. ACM, 1991.

[29] B. c. Kwon, W. Javed, N. Elmqvist, and J. S. Yi. Direct manipulation through surrogate objects. In *Proc. CHI '11*, pages 627–636. ACM, 2011.

[30] Y. Luo and D. Vogel. Crossing-based selection with direct touch input. In *Proc. CHI '14*, pages 2627–2636. ACM, 2014.

[31] M. J. McGuffin, N. Burtnyk, and G. Kurtenbach. Fast sliders: Integrating marking menus and the adjustment of continuous values. In *Proc. GI '02*, pages 35–41. Canadian Information Processing Society, 2002.

[32] T. Moscovich. Contact area interaction with sliding widgets. In *Proc. UIST '09*, pages 13–22. ACM, 2009.

[33] C. Perin and P. Dragicevic. Manipulating multiple sliders by crossing. In *Proc. IHM '14*, pages 48–54. ACM, 2014.

[34] C. Perin, P. Dragicevic, and J.-D. Fekete. Revisiting bertin matrices: New interactions for crafting tabular visualizations. *IEEE TVCG*, 20(12):2082–2091, Dec 2014.

[35] C. Perin, F. Vernier, and J.-D. Fekete. Interactive horizon graphs: Improving the compact visualization of multiple time series. In *Proc. CHI '13*, pages 3217–3226. ACM, 2013.

[36] R. Raisamo and K.-J. Räihä. A new direct manipulation technique for aligning objects in drawing programs. In *Proc. UIST '96*, pages 157–164. ACM, 1996.

[37] R. Rao and S. K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proc. CHI '94*, pages 318–322. ACM, 1994.

[38] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3):237–256, 1982.

[39] B. Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69, 1983.

[40] M. Spenke, C. Beilken, and T. Berlage. Focus: The interactive table for product comparison and selection. In *Proc. UIST '96*, pages 41–50. ACM, 1996.

[41] R. St. Amant and T. E. Horton. Characterizing tool use in an interactive drawing environment. In *Proc. SMARTGRAPH '02*, pages 86–93. ACM, 2002.

[42] A. Van Dam. Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67, 1997.

[43] Wikipedia. Equalization (audio), 2014. `http://en.wikipedia.org/wiki/Equalization_(audio)`.