Manipulation de Sliders Multiples par Franchissement

Charles Perin, Pierre Dragicevic

▶ **To cite this version:**

Charles Perin, Pierre Dragicevic. Manipulation de Sliders Multiples par Franchissement. IHM'14, 26e conférence francophone sur l'Interaction Homme-Machine, Oct 2014, Lille, France. pp.48-54, 10.1145/2670444.2670449 . hal-01089629

HAL Id: hal-01089629
https://hal.science/hal-01089629

Submitted on 2 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Manipulation de Sliders Multiples par Franchissement

Charles Perin
INRIA, Univ. Paris-Sud,
CNRS-LIMSI
91400, Orsay, France
charles.perin@inria.fr

Pierre Dragicevic
INRIA
91400, Orsay, France
pierre.dragicevic@inria.fr

RÉSUMÉ
Le crosset est un nouvel instrument d'interaction basé sur le franchissement et exploitant la dimension orthogonale des widgets. Nous explorons l'espace de conception des crossets et illustrons leur utilisation par un cas d'application : l'exploration visuelle et le formatage de tables numériques. À notre connaissance, le franchissement n'a jamais été appliqué à la manipulation simultanée de sliders multiples ni à l'interaction avec des tables. Cet article ouvre de nombreuses perspectives et nous espérons voir plus d'interfaces basées sur cette technique dans le futur.

Mots Clés
Manipulation Directe ; Interaction Instrumentale ; Sliders ; Franchissement ; Visualisation.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous.

INTRODUCTION

Cet article présente le crosset (crossing widget), un widget franchissable pour la manipulation simultanée de sliders multiples. Par slider, nous entendons les sliders ainsi que les widgets dérivés tels que les sliders à plusieurs poignées, les boutons radio et les cases à cocher, comme nous le détaillons par la suite (Figure 4). La conception du crosset est basée sur les principes de la manipulation directe [23] et sur le modèle de l'interaction instrumentale [6, 7, 8].

Alors que les sliders sont uni-dimensionnels, des travaux ont exploité la dimension orthogonale au slider [4, 14]. Cependant, cette seconde dimension n'a jamais bénéficié du franchissement [1] (sélection de plusieurs objets par un geste unique) alors que l'interaction est adaptée à la manipulation de sliders multiples. Contrairement au pointage qui requiert de déplacer le curseur à l'intérieur d'un objet graphique et presser le bouton de la souris pour l'activer, le franchissement consiste à déplacer le curseur sur une série d'objets graphiques pour les activer.

Nous décrivons les propriétés du crosset et montrons que toute une hiérarchie de widgets peuvent hériter de ces propriétés. Nous explorons l'espace de conception des crossets, en nous basant sur une caractérisation des widgets

© ACM, 2014. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Actes de la 26ième conférence francophone sur l'Interaction Homme-Machine, 2014.
http://dx.doi.org/10.1145/2670444.2670449

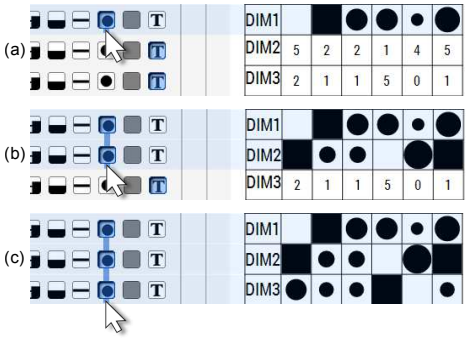

Figure 1. Illustration du franchissement de crossets radio. Les boutons permettent de sélectionner l'encodage visuel des lignes. Les lignes sont progressivement encodées par franchissement.

dérivés du slider continu standard. Nous appliquons les crossets à un projet récent [21] pour l'exploration visuelle interactive de tables numériques (Figure 1). Les crossets sont compatibles avec l'agencement des tables, génériques et polyvalents, et simples d'utilisation.

CONTEXTE

Les sliders ont des fonctionnalités multiples. Ils sont par exemple présents dans les lecteurs vidéo pour la navigation temporelle et dans les fenêtres pour faire défiler le contenu d'une page. Les sliders sont aussi souvent utilisés pour attribuer une valeur à un paramètre dans un intervalle de valeurs possibles, par exemple pour régler l'opacité d'un calque entre 0 et 100 dans un éditeur de photos.

Dimension Orthogonale

Le *degré d'intégration* [6, 7] est le rapport entre le nombre de degrés de liberté utilisés pour manipuler un instrument et le nombre de degrés de liberté de l'instrument physique (*e. g.,* la souris) utilisé pour manipuler cet instrument. Un slider possédant un axe, il ne prend en compte qu'une dimension de l'espace et a un degré d'intégration de 1/2. Des techniques utilisent la dimension orthogonale du slider, résultant en un degré d'intégration de 2/2 [4, 14]. Cette seconde dimension est utilisée pour modifier la granularité de la navigation et augmenter la précision.

L'Infovis toolkit [14] permet de créer des sliders multi-résolution dont la précision dépend de la distance orthogonale. Certaines versions de Quicktime proposent aussi cette fonctionnalité, où la granularité de la barre de navigation temporelle est ajustée en fonction de la distance au slider.

Orthozoom [4] étend la barre de défilement classique en un instrument intégrant à la fois le pan et le zoom : déplacer la poignée le long de l'axe du slider fait défiler la page, et la dimension orthogonale est utilisée pour zoomer dans la page en fonction de la distance orthogonale au slider.

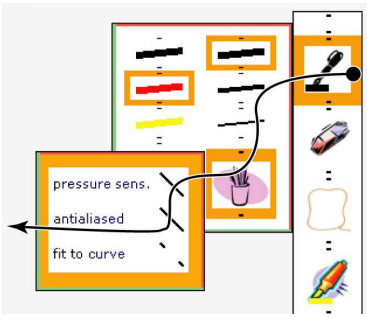

Figure 2. Exemple de geste de franchissement dans CrossY. Un geste unique permet de sélectionner successivement l'outil de dessin, le type de trait, et l'anti-aliasing.

Malgré l'existence de ces techniques, les sliders sont généralement standardisés et uni-dimensionnels. Il est en revanche fréquent de trouver des séries de sliders alignés.

Sliders Multiples

L'exemple fondateur en Infovis des sliders multiples est les requêtes dynamiques [2], pour l'exploration de données multi-dimensionnelles. Cette approche est aujourd'hui démocratisée et les requêtes dynamiques sont omniprésentes, en particulier sur les sites marchands.

Paramétrer plusieurs sliders en les modifiant un par un a du sens pour attribuer une valeur différente à chaque slider, mais est peu efficace pour donner la même valeur à plusieurs sliders. Par exemple, il est fréquent de vouloir réinitialiser l'état des sliders en leur attribuant la valeur minimale ou maximale possible. Il n'existe actuellement pas de technique permettant cette interaction alors que la dimension orthogonale du slider est inutilisée.

Franchissement

Bien que le franchissement ait été proposé en 2002 [1], très peu d'interfaces utilisent cette technique malgré plusieurs avantages par rapport au pointage, le principal étant la possibilité de "franchir" plusieurs objets d'un même geste, par exemple une série de sliders pour modifier leur valeur.

Peu de travaux sont basés sur cette technique. Dragicevic utilise le franchissement pour glisser-déposer un objet dans une fenêtre en arrière plan [13], permettant d'interagir avec un objet secondaire tout en manipulant un objet d'intérêt.

La Figure 2 illustre CrossY [3], une interface de dessin basée sur le franchissement. CrossY permet d'effectuer une sélection d'éléments dans des menus successifs en parcourant l'arborescence d'un geste continu au lieu de cliquer sur chaque élément des menus et sous-menus.

Baudisch explore les vertus du franchissement pour les interfaces contenant de nombreux boutons à deux états [5], et les Sliding Widgets [20] permettent de manipuler des widgets classiques gestuellement. En particulier, ils permettent d'interagir avec plusieurs cases à cocher simultanément par un geste unique. Cependant, ce cas reste marginal puisque les Sliding Widgets sont conçus pour manipuler un widget unique. L'auteur aborde néanmoins la manipulation simultanée de deux sliders en utilisant une aire d'activation lors du pointage, mais sans considérer la dimension orthogonale comme moyen de sélection des widgets.

Finalement, Accot et al. explorent les propriétés du franchissement [1]. Ils montrent que le franchissement effectué perpendiculairement à la forme de l'objet et de manière continue est le plus efficace. En revanche, le franchissement requiert de contrôler la direction du curseur en permanence afin de cibler les objets à franchir (voir Figure 2).

Généralisation

Les sliders alignés (disposition en grille) sont compatibles avec les tableurs (*e. g.,* Excel et Google Spreadsheet) qui permettent d'interagir avec des lignes et des colonnes (*e. g.,* sélection, tri, redimensionnement). Excel propose au moins deux manières de redimensionner des colonnes. Après sélection des colonnes à redimensionner, un clic droit ouvre un menu contextuel affichant la commande, puis une boîte de dialogue permet de spécifier au clavier la largeur des colonnes en centimètres. Le même résultat peut être obtenu en agrandissant une des colonnes sélectionnées par glisser-déposer de sa bordure. Bien que la seconde manière soit plus directe spatialement [7], le retour visuel est difficile à interpréter car l'aperçu du redimensionnement n'est disponible que pour la colonne manipulée et l'action n'est effectuée que lorsque le bouton de la souris est relâché. De plus, il est nécessaire dans les deux cas d'effectuer une séquence d'actions (sélection puis redimensionnement), impliquant une faible directitude temporelle [7].

L'interaction avec un ensemble de calques dans les logiciels tels que Photoshop illustre le besoin d'attribuer à plusieurs éléments adjacents et alignés une même commande. Photoshop permet d'ailleurs d'afficher/masquer une série de calques en maintenant le bouton de la souris pressé sur l'icône en forme d'œil et en faisant glisser la souris sur d'autres lignes de calques. Cependant, l'action n'est pas réversible et requiert toujours d'effectuer autant de tâches de pointage qu'il y a d'icônes à traverser. De plus, si le franchissement est possible pour la visibilité des calques (valeur binaire), elle n'est pas disponible par exemple pour régler l'opacité des calques (valeur continue).

Problème

Pour résumer, le slider est un widget standard largement utilisé, mais souffre des points suivants :

1. Le slider standard n'utilise qu'une dimension de l'espace alors que la dimension orthogonale est disponible.
2. Les sliders multiples sont fréquents, et manipuler plusieurs sliders individuellement peut être fastidieux.
3. L'alignement des sliders multiples n'est pas exploité.
4. Le franchissement peut apporter une solution, mais le geste associé requiert une lourde charge cognitive dû au pointage permanent en deux dimensions.

D'une part, le franchissement est optimal lorsque la trajectoire est orthogonale à l'objet à franchir, et contraindre l'interprétation du geste tout en laissant le geste de l'utilisateur libre requiert une faible charge cognitive et empêche les erreurs [1]. D'autre part, la dimension orthogonale, qui intersecte l'ensemble des sliders multiples lorsqu'ils sont disposés sous forme de grille, n'est pas utilisée. En conséquence, notre approche est basée sur le franchissement avec interprétation du geste contraint orthogonalement au widget.

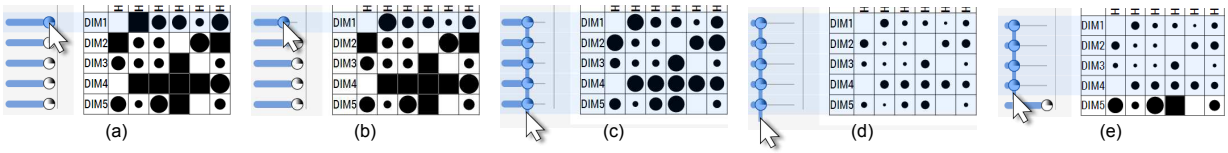

Figure 3. Franchissement de crossets (sliders continus).

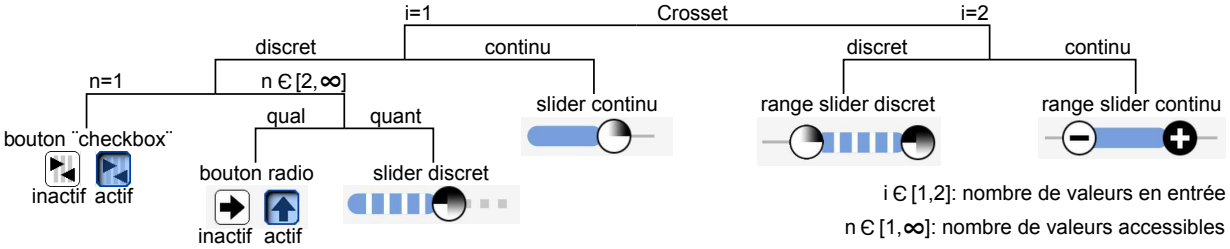

Figure 4. Les différentes implémentations de crossets.

CROSSET : WIDGET FRANCHISSABLE

Le crosset est un widget augmenté de capacités de franchissement afin de modifier simultanément des widgets adjacents. La conception des crossets détaillée ici s'applique aux interfaces en grille (tableurs, matrices, tables).

La Figure 3 illustre la technique, où les crossets (ici, des sliders continus) placés en face des lignes d'une table affectent l'encodage visuel des lignes : (a) le franchissement est activé par sélection de la poignée d'un slider ; (b) déplacer la souris horizontalement permet de naviguer dans le domaine des valeurs, comme pour un slider classique, et l'encodage visuel de la ligne associée est mis à jour ; (c) déplacer la souris verticalement permet de naviguer dans le domaine de la sélection : les crossets franchis prennent la valeur correspondant à la position horizontale de la souris ; (d) déplacer la souris horizontalement modifie la valeur des crossets franchis ; et (e) désélectionner un crosset en le franchissant à nouveau restaure sa valeur avant sélection.

La manipulation des crossets comporte trois étapes :

1. **Début :** le franchissement est initié par une pression du bouton de la souris sur la poignée/le bouton du crosset. La poignée/bouton change alors de représentation visuelle pour signaler à l'utilisateur sa sélection.
2. **Geste :** déplacer la souris dans l'espace permet de naviguer à la fois dans le domaine des valeurs et le domaine de la sélection. La représentation graphique du crosset est mise à jour afin de fournir un retour immédiat à l'utilisateur. De plus, une ligne orthogonale au slider reliant le slider au curseur indique le franchissement en cours.
3. **Fin :** le franchissement se termine lorsque le bouton de la souris est relâché. Des modifications peuvent s'appliquer à cette étape, comme nous le détaillons plus loin.

Types de crossets

Le crosset est générique dans le sens où il permet d'augmenter une variété de widgets standards. La Figure 4 illustre les crossets implémentés. Un crosset dépend de deux paramètres : i est le nombre de variables manipulables par l'utilisateur en entrée. Les cas les plus courants sont $i = 1$ et $i = 2$, mais Photoshop par exemple dispose de sliders à autant de poignées que désiré pour définir des gradients de couleur ; et $n \in [1, \infty]$ est le nombre de valeurs accessibles (domaine des valeurs du crosset). Puisque le franchissement ne fait pas partie des implémentations standard, nous avons dû créer une librairie (Javascript et d3 [11]) implémentant ces widgets à partir de zéro.

1. Le widget de base est le slider continu ($i = 1, n > 1$). Il se comporte comme un slider classique.
2. Le slider discret est un slider continu où les valeurs possibles sont magnétisées. La représentation visuelle du crosset illustre ces valeurs discrètes.
3. Le range slider continu ($i = 2, n > 1$) est un slider continu à deux poignées pour spécifier une valeur min et une valeur max. L'implémentation requiert de spécifier un écart minimal entre le min et le max afin de garantir la sélection des poignées et assurer $min < max$.
4. Le range slider discret est un cas particulier du range slider continu, de la même manière que le slider discret est un cas particulier du slider continu. Les mêmes considérations s'appliquent pour l'implémentation.
5. Le bouton radio est similaire au slider discret ($i = 1, n > 1$) mais est dédié aux valeurs non ordinales. Chaque bouton peut être vu comme un pas de slider discret et la sélection est exclusive (illustré Figure 1). Ce crosset ne comporte pas d'axe reliant les boutons et deux états permettent d'identifier le bouton activé.
6. Le bouton checkbox ($i = 1, n = 1$) peut être soit actif, soit inactif. Puisque $n = 1$, le déplacement dans le domaine des valeurs est inutile. Cependant, le domaine des valeurs pourrait déclencher des actions supplémentaires, telles que l'activation et la désactivation.

Propriétés des crossets

Le crosset respecte de nombreux principes issus de la manipulation directe et de l'interaction instrumentale.

Cohérence interne. Le crosset est un instrument générique. Il assure donc la cohérence interne de l'interface, en termes de design, propriétés graphiques, couleurs, etc. [18].

Cohérence externe. C'est la cohérence d'une interface par rapport aux interfaces existantes. Elle favorise l'apprentissage rapide et le transfert de connaissances [18, 19, 22]. Les crossets ont une grande cohérence externe puisqu'ils sont des widgets augmentés : les crossets disposent des interactions des widgets classiques. En particulier, le crosset reste compatible avec le clic (**Début** suivi immédiatement de la **Fin**, sans considérer le **Geste**). Le franchissement n'est

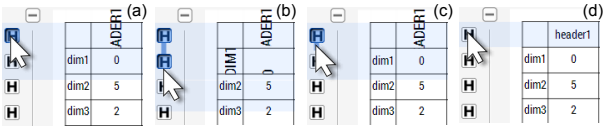

Figure 5. Après le franchissement initial, entrer à nouveau dans le premier crosset revient à annuler le geste en cours.

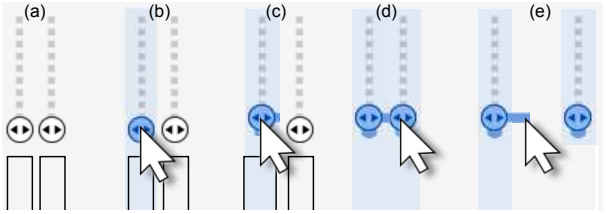

Figure 6. Problème lorsque l'action associée à un crosset modifie la position d'un ou plusieurs crossets.

enclenché que lorsque l'utilisateur en exprime le besoin en déviant orthogonalement à l'axe du widget.

Degré d'intégration. Contrairement au widget standard, qui a un degré d'intégration de 0/2 (pour les boutons) ou 1/2 (pour les sliders), les crossets ont un degré d'intégration de 2/2 puisque les deux dimensions de la souris sont utilisées.

Actions immédiates et réversibles. La représentation graphique du crosset est mise à jour afin de fournir un retour visuel immédiat [23]. De plus, le résultat est immédiat et réversible [23], comme illustré sur la Figure 3(d-e). La Figure 5 montre la réversibilité du geste pour le crosset checkbox : (a) un bouton est activé ; (b) un second bouton est activé ; (c) ce second bouton est désactivé ; (d) le premier bouton est désactivé, restaurant l'état initial.

Multi-Cibles. Le geste permet d'appliquer une action à un ensemble d'objets, ce qui est un des défis des interfaces à manipulation directe [15, 16, 17]

Décalage temporel. Puisque la sélection des objets d'intérêt et la manipulation des valeurs se fait d'un geste unique avec retour visuel immédiat, la technique minimise le décalage temporel [7] et la distance articulatoire [19].

Geste contraint. Le geste de franchissement est *contraint*. Puisqu'une seule des deux dimensions est réservée au franchissement, le geste est libre mais son interprétation par le système est verrouillée dans la direction orthogonale au crosset. Cette propriété garantit que l'utilisateur ne commettra pas d'erreur de sélection [1], sans nécessiter de message intrusif interférant avec la tâche en cours [19, 22].

Actions modifiant l'agencement des crossets

Lorsque l'action associée à un crosset affecte l'agencement d'un ou plusieurs crossets, il est parfois impossible d'appliquer l'action immédiatement. Prenons l'exemple du redimensionnement de colonnes par sliders illustré sur la Figure 6 : modifier la valeur du premier crosset n'affecte pas la position du second (b-c) ; mais franchir le second crosset provoque son déplacement vers la droite (d-e), et sa désélection. Le crosset reprend alors sa valeur avant sélection et sa position initiale... et est à nouveau franchi immédiatement, puisque sous le pointeur. En résulte une succession indésirable de va-et-vient très rapides du crosset.

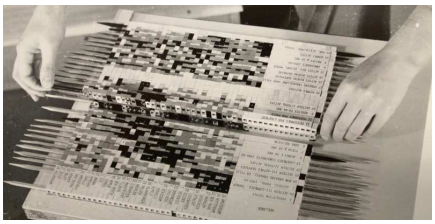

Figure 7. Manipulation d'une des matrices physiques de Bertin.

Les crossets affectant la disposition de crossets requièrent donc une stratégie différente. Notre approche est similaire au déplacement d'une fenêtre sous Windows quand les performances de l'ordinateur sont limitées, où seul le cadre de la fenêtre est affiché pendant le déplacement et la fenêtre est actualisée complètement lorsque la souris est relâchée. De la même manière, nous donnons un aperçu partiel du résultat et n'appliquons les modifications de disposition qu'au relâchement de la souris. Cela assure la compatibilité des crossets pour ce genre d'opérations, au détriment d'une réduction de la directitude temporelle de l'instrument [7].

CONCEPTION D'UNE INTERFACE BASÉE CROSSETS

Nous avons appliqué les crossets à un outil de formatage de table basé sur les matrices de Bertin [9], un dispositif physique illustré Figure 7. Bertifier [21] est une interface web reproduisant cette méthode d'analyse et exploration de données tabulaires (Figure 8, **`www.bertifier.fr`**). Cette méthode implique de nombreuses étapes, telles que le conditionnement des données, le choix de l'encodage visuel, et le réordonnancement des lignes et des colonnes.

Après avoir identifié les fonctionnalités requises, nous avons été confrontés à leur hétérogénéité. Une approche naïve aurait été de concevoir une interface WIMP proposant des boîtes de dialogue successives, comme cela a déjà été proposé [10, 12, 25]. Cependant, aucune de ces interfaces n'est parvenue à démocratiser la méthode de Bertin. Nous pensons que cela est principalement dû à la mauvaise conception des interactions, résultant en des interfaces puissantes mais particulièrement complexes à apprendre et utiliser. En revanche, certaines implémentations proposent moins de fonctionnalités mais restaurent la directitude des actions [24], s'inspirant de la méthode physique. Néanmoins, il n'existe aucune interface reproduisant cette méthode—malgré des tentatives pendant plus de 30 ans—qui soit à la fois puissante, simple et cohérente. C'est pour palier ce manque que nous avons conçu Bertifier.

Disposition des crossets

Une table est constituée de lignes et colonnes. L'objet d'intérêt peut être la cellule, comme c'est le cas pour les tableurs. Mais la méthode de Bertin spécifie que les objets d'intérêt sont les lignes et les colonnes, les actions de l'utilisateur doivent donc s'appliquer à ces objets.

Une table étant rectangulaire, il est pertinent de placer les commandes s'appliquant aux lignes et aux colonnes respectivement à gauche et au-dessus de la table. Au lieu d'adopter une démarche séquentielle 1) sélection des lignes/colonnes, 2) choix de l'action à appliquer par le biais d'un menu externe, nous dupliquons chaque action pour chaque ligne/colonne et maximisons ainsi la directitude

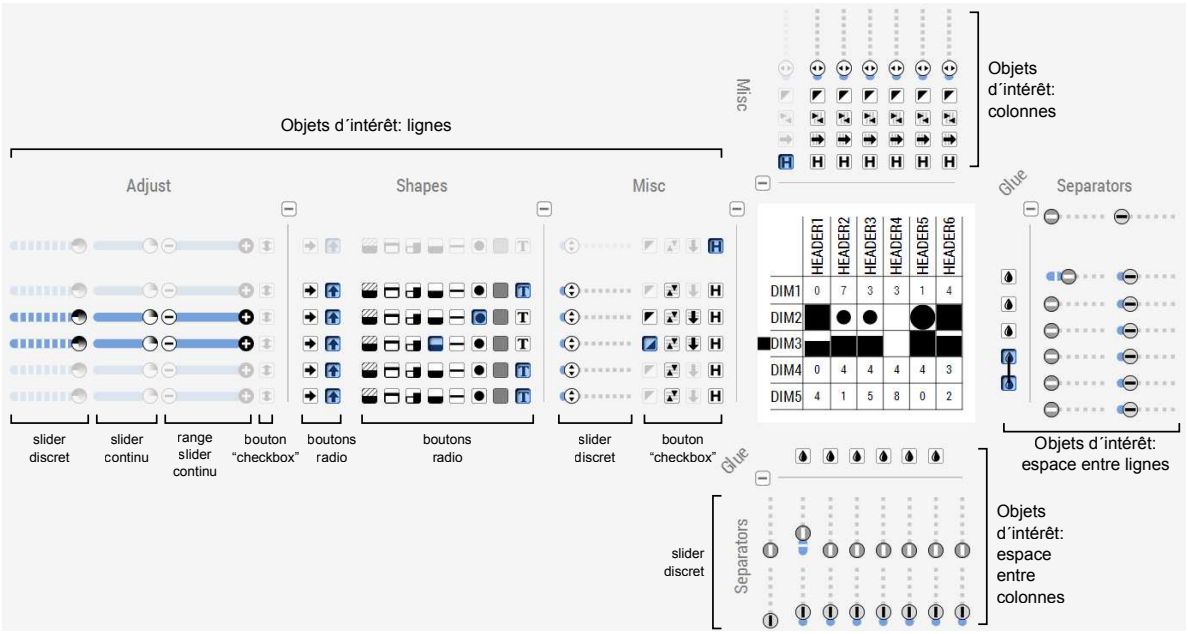

Figure 8. Interface de Bertifier. Les crossets sont disposés autour de la table.

spatiale [7]. Lors de notre étude de la méthode de Bertin, nous avons réalisé que certaines commandes ne s'appliquaient pas aux lignes/colonnes, mais aux espaces entre celles-ci ; c'est le cas des séparateurs permettant d'identifier des groupes de lignes/colonnes visuellement. Puisqu'une table a quatre côtés, nous utilisons donc l'espace à droite et en-dessous de la table pour y placer les crossets visant l'espace entre les lignes et les colonnes, respectivement. Au final, les crossets sont alignés en quatre grilles et compatibles avec le franchissement orthogonal.

Contrairement aux interactions standard des tableurs, la sélection des lignes/colonnes peut être modifiée à la volée. Afin de faciliter la correspondance entre un crosset et son objet d'intérêt, nous ajoutons une surbrillance aux crossets ainsi qu'à l'objet d'intérêt cible lorsque la souris est positionnée sur un groupe de crossets. Enfin, les crossets sont regroupés en catégories expansibles afin de ne pas surcharger l'interface et de maximiser la directitude spatiale.

De plus, le positionnement des crossets autour de la table assure la visibilité et l'accessibilité des cellules. Grâce à cela, nous avons pu implémenter le glisser-déposer de lignes/colonnes et conserver la métaphore inspirée de la manipulation physique des matrices de Bertin.

Actions modifiant l'agencement spatial de la table

Nous avons évoqué les actions modifiant la disposition des crossets. Dans Bertifier, trois actions sont concernées (illustrées Figure 9) et disposent d'un retour visuel partiel.

L'encodage visuel des cellules d'une ligne/colonne en texte affecte la largeur et/ou la hauteur de la cellule. L'aperçu partiel peut consister en une rotation du texte, une écriture en lettres capitales ou minuscules, et une modification de l'ancre du texte (a). Puisque les lignes/colonnes ne sont pas modifiées immédiatement, le texte peut déborder de la cellule. Quand le geste est terminé, les lignes/colonnes sont animées vers leur nouvelle position/taille (b).

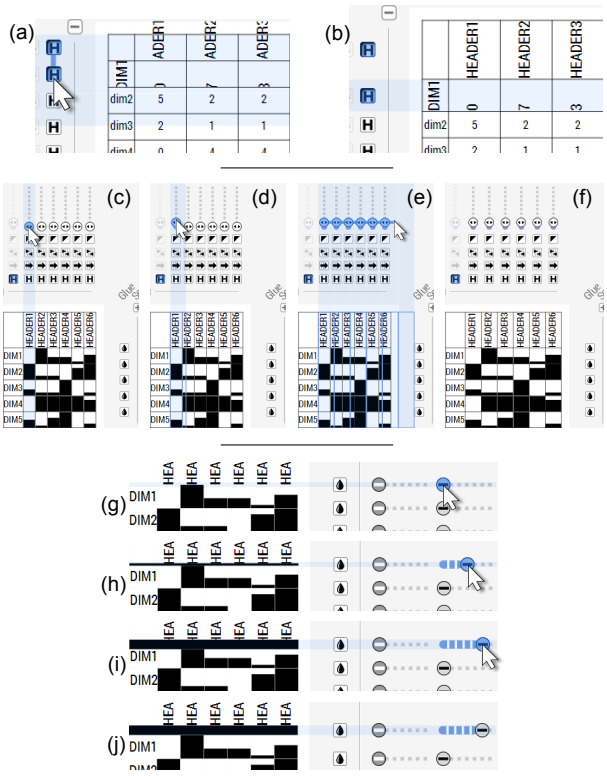

Figure 9. Trois exemples d'aperçu immédiat du résultat du franchissement où la disposition des crossets n'est mise à jour qu'au relâchement du bouton de la souris.

Le redimensionnement de lignes et colonnes affecte leur hauteur et largeur, respectivement. Dans le cas d'une colonne, après sélection d'un crosset (c), déplacer verticalement la poignée change la largeur de la colonne. Nous affichons le cadre qu'auraient les colonnes si le bouton de la souris était relâché comme aperçu, afin de ne pas déplacer les crossets lors du franchissement (d-e). Lorsque le geste se termine, les colonnes sont animées vers leur nouvelle position/taille (f).

La modification de séparateurs affecte la position des lignes/colonnes. Changer la valeur d'un slider discret (g) donne un aperçu du séparateur modifié. Puisque les lignes/colonnes ne sont pas déplacées immédiatement, le séparateur s'affiche par dessus celles-ci (h-i). Comme précédemment, les lignes/colonnes sont animées vers leur nouvelle position une fois le geste terminé (j).

DISCUSSION ET LIMITATIONS

Un point crucial lors de la conception des crossets a été la décision de verrouiller la direction du franchissement perpendiculairement à l'axe du crosset. En effet, cette stratégie permet d'éviter un problème majeur des interfaces basées sur le franchissement qu'est le suivi de tunnels, une tâche moteur lente [1]. Avec les crossets, l'utilisateur peut dévier de la trajectoire "normale" sans risquer de sélectionner des crossets différents par erreur. Cela permet donc d'accélérer l'interaction en autorisant un geste approximatif sans risque d'erreur.

Limitations

Les crossets ont certaines limitations que nous discutons dans le reste de cette section.

Degré de compatibilité. Le degré de compatibilité mesure la similarité entre les actions physiques de l'utilisateur sur l'instrument et la réponse de l'instrument [7]. Les crossets ont un degré de directitude temporelle et spatiale et un degré d'intégration élevés, mais leur degré de compatibilité est discutable. Les boutons ont un degré de compatibilité élevé, mais pour certains sliders—redimensionnement des lignes et colonnes, création de séparateurs—le geste de l'utilisateur est orthogonal au résultat. Par exemple, pour augmenter/diminuer la hauteur d'une ligne, la poignée est déplacée vers la droite/gauche. Déplacer la poignée vers le haut/bas serait plus compatible avec l'action associée au slider, mais en conflit avec la dimension de sélection des crossets du geste de franchissement.

Espace Écran. Les crossets doivent être visibles à l'écran pour être franchis. Pour notre cas d'application, cela peut s'avérer problématique si la table est trop grande pour être affichée entièrement à l'écran, forçant l'utilisateur à effectuer une série de gestes couplée à un déplacement dans l'espace de travail. Cependant, le zoom résout—dans beaucoup de cas—ce problème. Une perspective prometteuse pour réduire l'espace écran nécessaire serait de rendre les crossets transients afin de n'apparaître qu'à l'activation du geste et de disparaître une fois le geste terminé.

Crossets Non-Adjacents. L'espace de conception des crossets que nous proposons s'applique à toute interface en grille, et la généricité des crossets se restreint à ces (nombreuses) interfaces : il est nécessaire que les crossets soient adjacents et les dimensions similaires. Un problème est en conséquence l'attribution d'une même valeur à des crossets non adjacents. Puisque la direction du franchissement est orthogonale aux crossets, il est impossible d'éviter un crosset. Il serait envisageable d'utiliser un modificateur clavier pour "éviter" un crosset, mais cela ajouterait de la complexité à l'interaction dont l'un des atouts majeurs est la simplicité. Nous avons contourné le problème par les crossets discrets où le faible nombre de valeurs possibles permet d'attribuer visuellement la même valeur à des crossets éloignés. Dans le cas des tables de Bertin, le problème peut aussi être contourné en déplaçant les lignes/colonnes par glisser-déposer afin de les disposer côte à côte. Nous pensons cependant que d'autres stratégies pourraient s'avérer pertinentes, et ceci fait partie des travaux futurs que nous envisageons.

Stabilité de l'Interface. Enfin, nous avons illustré le problème de stabilité de l'interface à travers l'exemple du redimensionnement de colonnes. En particulier, le franchissement peut être perturbé lorsque l'action en cours affecte l'agencement spatial de l'interface.

Généralisation

Nous avons appliqué les crossets aux tables mais de nombreuses applications bénéficieraient de widgets franchissables. C'est le cas des requêtes dynamiques [2], où plusieurs dimensions sont manipulées. Par exemple, lors d'une recherche d'appartement, modifier simultanément la distance à A et la distance à B afin de spécifier que les deux distances ont la même importance est bénéfique ; ou, manipuler simultanément les trois sliders des couleurs dans l'espace RGB permet de spécifier un niveau de gris. L'implémentation actuelle est dédiée à des requêtes de même nature. Cependant, dans le cas fréquent où les dimensions sont différentes, attribuer la même valeur à des crossets est peu pertinent. Néanmoins, les dimensions sont souvent normalisées, permettant d'effectuer des requêtes similaires sur des sliders de sémantique différente (*e. g.,* les indicateurs hétérogènes du Better Life Index sont exprimés entre 0 et 1). Au contraire, parfois les dimensions sont les mêmes mais leur assigner des valeurs identiques est peu souhaitable : tout dépend de la sémantique des données.

Enfin, une perspective prometteuse serait d'autoriser des gestes en 2D non contraints : effectuer une requête consisterait à dessiner une trajectoire en 2D spécifiant en un geste unique un ensemble de valeurs pour des dimensions différentes. Se pose alors la question de l'activation du mode, qui risque d'ajouter de la complexité à l'interaction.

CONCLUSION

Nous avons proposé le crosset, un nouvel instrument d'interaction franchissable pour la manipulation simultanée de widgets multiples. Nous avons détaillé l'espace de conception des crossets et proposé une série d'implémentations illustrées par un cas d'application : la méthode de Bertin [9] pour l'exploration visuelle de tables numériques.

À notre connaissance, le franchissement n'a jamais été appliqué à la manipulation de sliders multiples simultanément ni pour interagir avec des tables. Cet article ouvre des perspectives prometteuses, et nous espérons voir plus d'interfaces basées sur cette technique dans le futur.

Puisque ce travail a nécessité une implémentation à partir de zéro, nous prévoyons de créer une librairie Javascript en open source. Nous espérons que cette librairie permettra à des concepteurs de sites web, de visualisation, ou d'IHM en général, de tirer profit des crossets et d'étendre leurs fonctionnalités.

BIBLIOGRAPHIE

1. Accot J. & Zhai S. More than dotting the i's — foundations for crossing-based interfaces. In *Proc. CHI'02*, ACM (2002), 73–80.
2. Ahlberg C. & Shneiderman B. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proc. CHI '94*, ACM (1994), 313–317.
3. Apitz G. & Guimbretière F. Crossy: A crossing-based drawing application. In *Proc. UIST'04*, ACM (2004), 3–12.
4. Appert C. & Fekete J.-D. Orthozoom scroller: 1d multi-scale navigation. In *Proc. CHI '06*, ACM (2006), 21–30.
5. Baudisch P. Don't click, paint! using toggle maps to manipulate sets of toggle switches. In *Proc. UIST '98*, ACM (1998), 65–66.
6. Beaudouin-Lafon M. Interaction instrumentale: de la manipulation directe à la réalité augmentée. In *Proc. IHM '97* (1997).
7. Beaudouin-Lafon M. Instrumental interaction: An interaction model for designing post-wimp user interfaces. In *Proc. CHI '00*, ACM (2000), 446–453.
8. Beaudouin-Lafon M. & Mackay W. E. Reification, polymorphism and reuse: Three principles for designing visual interfaces. In *Proc. AVI '00*, ACM (2000), 102–109.
9. Bertin J. *La graphique et le traitement graphique de l'information*. Nouvelle bibliothèque scientifique. Flammarion, 1975.
10. Bertin J. & Chauchat J.-H. Logiciel amado (analyse graphique d'une matrice de données). *CISIA, France* (1994).
11. Bostock M., Ogievetsky V. & Heer J. D3 data-driven documents. *IEEE TVCG 17*, 12 (2011), 2301–2309.
12. Caraux G. & Pinloche S. Permutmatrix: a graphical environment to arrange gene expression profiles in optimal linear order. *Bioinformatics 21*, 7 (2005), 1280–1281.
13. Dragicevic P. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proc. UIST '04*, ACM (2004), 193–196.
14. Fekete J.-D. The infovis toolkit. In *Proc. INFOVIS '04*, IEEE (2004), 167–174.
15. Frohlich D. M. The history and future of direct manipulation. *Behaviour & Information Technology 12*, 6 (1993), 315–329.
16. Frohlich D. M. Direct manipulation and other lessons. In *Handbook of human-computer interaction (2nd ed)*, Elsevier (1997), 463–488.
17. Gentner D. & Nielsen J. The anti-mac interface. *Commun. ACM 39*, 8 (1996), 70–82.
18. Grudin J. The case against user interface consistency. *Commun. ACM 32*, 10 (1989), 1164–1173.
19. Hutchins E. L., Hollan J. D. & Norman D. A. Direct manipulation interfaces. *Hum.-Comput. Interact. 1*, 4 (1985), 311–338.
20. Moscovich T. Contact area interaction with sliding widgets. In *Proc. UIST '09*, ACM (2009), 13–22.
21. Perin C., Dragicevic P. & Fekete J.-D. Revisiting bertin matrices: New interactions for crafting tabular visualizations. *IEEE TVCG* (2014).
22. Shneiderman B. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology 1*, 3 (1982), 237–256.
23. Shneiderman B. Direct manipulation: A step beyond programming languages. *Computer 16*, 8 (1983), 57–69.
24. Siirtola H. Interaction with the reorderable matrix. In *IV*, IEEE (1999), 272–279.
25. Wu H.-M., Tien Y.-J. & Chen C.-h. Gap: A graphical environment for matrix visualization and cluster analysis. *Comput. Stat. Data Anal. 54*, 3 (2010), 767–778.