1998 IEEE Symposium on Visual Languages, Halifax, Nova Scotia, September 1998, p. 118-125.

Fluid Visualization of Spreadsheet Structures

Takeo Igarashi *
Dept. of Info. Engineering
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
takeo@mtl.t.u-tokyo.ac.jp

Jock D. Mackinlay, Bay-Wei Chang, Polle T. Zellweger
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304, USA
{mackinlay, bchang, zellweger}@parc.xerox.com

Abstract

Spreadsheets augment a visible tabular layout with invisible formulas. Direct manipulations of the tabular layout may or may not result in the desired changes to the formulas. The user is forced to explore the individual cells to find, verify, and modify the formulas, which causes heavy cognitive overhead. We present a set of techniques that make these formulas and their resulting dataflow structure easily accessible while maintaining the natural appearance of the spreadsheet. ***Transient local views*** *visualize dataflow structures associated with individual cells, while* ***static global views*** *and* ***animated global explanations*** *visually present the entire dataflow structure at once.* ***Semantic navigation*** *enables the user to navigate through the dataflow structure interactively, and* ***visual editing*** *techniques make it possible to construct formulas using graphical editing techniques. Central to these techniques is the use of animation and lightweight interaction for rapid and non-intrusive visualization. Our prototype implementation suggests that these techniques can greatly improve the expressive power of current spreadsheets as well as other applications that have rich underlying structures.*

1 Introduction

Spreadsheets are one of the most successful applications making use of visual language techniques. They provide tabular layouts for displaying and manipulating complex information, and powerful visual programming mechanisms for applying dynamic structure to the static visual representation (Figure 1). Users can structure their problems in a spreadsheet by placing formulas in cells, automating the calculation of large complicated systems relatively easily.

However, access to these formulas and their resulting dataflow structure is not easy in current spreadsheet applications, resulting in significant cognitive overhead.

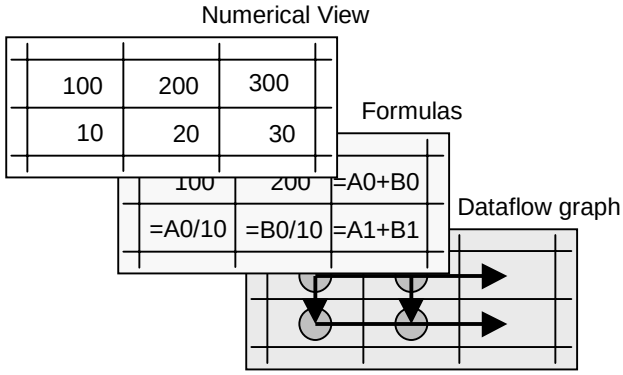

Figure 1. ***Dataflow graph of a spreadsheet.*** **A spreadsheet has an underlying dataflow graph in addition to the superficial numerical view. It is difficult to understand the structure of the dataflow graph because it is usually invisible.**

Often the layout provides cues about the underlying dataflow structure of the spreadsheet, making it relatively easy to understand the structure of the dataflow graph: a summation formula is typically placed at the end of the corresponding row or column. But in some cases, a user must put formulas referring to distant cells in irregular positions, which makes it difficult to understand the structure. In these situations, the user is forced to explore the individual cells to find, verify, and modify the formulas.

An even trickier problem occurs when the user moves the cells of a spreadsheet with cut and paste or through direct manipulation. Current spreadsheet applications adjust the spreadsheet formulas to maintain the original semantics of the dataflow, which may or may not be what the user desires. For example, a user exploring various "what if" scenarios may move a line that participates in a sequence of numbers, intending to see the sequence without the number. However, when there are formulas involved, the spreadsheet application adjusts them to ensure that they still participate in the sequence, leading to an error in the intended "what if" calculation. Because the formulas are invisible, the user may well be unaware of this mismatch with his intentions.

* This work was done while the first author was a summer intern at Xerox PARC.

Sophisticated users realize that they must check formulas after every move operation to ensure that they are correct.

A related problem occurs when a formula is used to fill a region of a spreadsheet. Current spreadsheet applications adjust the cell addresses in the formulas by the distance from the source formula, which again may or may not be what the user desires. To inhibit these adjustments, the user can specify a cell in the formula to be an absolute reference (by using the "$" symbol), but it is difficult to place the $ symbol correctly, which means the filled formulas must also be checked to make sure they are correct [12].

The invisibility of formulas also causes trouble for users when they need to understand spreadsheets created by others. The user must repeatedly select a cell, read the formula, and move on to the next cell, until he has seen enough formulas to get an overview of the spreadsheet. As spreadsheets get larger and more complicated, the overhead of understanding shared spreadsheets increases dramatically.

In this paper we present a set of techniques to make implicit spreadsheet structures visible and easily accessible, by letting users interact directly with the visual dataflow structure instead of accessing the dataflow indirectly through textual formulas. We use graphical variation (color, shading, outlining, etc.), animation, and lightweight interaction to visualize underlying dataflow structures while minimizing clutter on the screen.

The first three techniques graphically augment formula cells to help users visually understand the dataflow structure without tediously clicking each cell and reading its formula. The basic idea is to render the dataflow graph directly on the screen without unduly interfering with the natural appearance of the spreadsheets. The **transient local view** allows users to view dataflow structures associated with an individual cell in a lightweight manner. The **static global view** visualizes the entire dataflow graph of a spreadsheet by overlaying the static graph on the tabular layout. The **animated global explanation** automatically generates and plays an animated presentation of the dataflow through the spreadsheet cells to describe structures that are difficult to see with the static overlay. **Semantic navigation** allows users to traverse the spreadsheet structure interactively based on logical connectivity in cell referencing. **Visual editing** techniques allow users to construct dataflow graphs efficiently using direct manipulation and programming by example. While semantic navigation and visual editing are not themselves visualizations, they greatly contribute to the *visual understanding* of the dataflow graph, augmenting the first three techniques.

We call these techniques *fluid* because of their interaction style and their visual representation: many interactions proceed in a smooth manner with the user expressing interest simply by moving the mouse, and the visualizations of the dataflow structure are gracefully integrated with the original tabular layout of the spreadsheet. Our approach is to augment the existing tabular display of the spreadsheet rather than replacing it with a separate visualization. This allows users to understand the formulas in the context of the regular spreadsheet layout.

We have implemented these interaction techniques in a prototype spreadsheet program, written using Pad++ [1, 23] and Python running on Unix platforms. Although the prototype system is limited in performance—animation becomes sluggish at spreadsheets larger than 20x20 cells—we observed that our fluid visualization techniques dramatically facilitated the understanding of the hidden spreadsheet structure.

The rest of the paper is organized as follows. First, we describe the current state of the art in spreadsheet visualization. Next we present our visualization techniques in detail. Finally, we discuss some related work and summarize the paper.

2 State of the art

Given the problems associated with invisible formulas, it is not surprising that various attempts have been made to make them more visible. For example, Microsoft Excel 97™ [18], arguably the most feature-rich spreadsheet application available, includes two techniques that provide limited visualization of the dataflow graph for a given cell. The first feature, called the "Range Finder," can be invoked by selecting a cell that has a formula and clicking in the formula bar. When this is done, the addresses in the formula are colored and the corresponding regions in the spreadsheet are surrounded with colored rectangles, which can be moved and adjusted to edit the formula in a direct manipulation fashion.

The second feature, "Auditing," draws arrows from the selected cell to its ancestors or descendants. In addition, the user can directly jump to an ancestor or descendant by double-clicking the arrowhead.

These techniques try to make the hidden dataflow structure visually accessible, but they must be invoked via menus or toolbar buttons, and they are limited to showing the dataflow for a single cell at a time. There is no way to display the overall structure of the spreadsheet. The user must click in each cell individually to see the Range Finder colored rectangles, and complicated spreadsheets can create a tangle of arrows, making it difficult to see the relationships among cells.

Like these parts of Excel, our work is focused on showing the dataflow structure. Unlike Excel, however, we are interested in showing both local portions of the structure and global views. In addition, we optimize the visualization based on the spatial relationships of the dataflow structure, and we use both transient techniques (see the next section) and animation to fluidly present the dataflow structure.

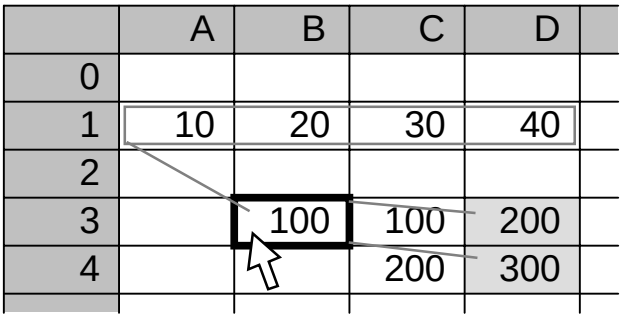

Figure 2. *Transient visualization of a cell*. The transient local view visualizes the incoming and outgoing cells of the current cell. Incoming cells are enclosed by thin lines, and outgoing cells are filled with gray. B3 refers to A1 through D1, and is referred to by D3 and D4.

3 Transient local view

The transient local view technique allows the user to see a part of a dataflow graph associated with the *current* cell, the cell that the user is interacting with. The system visualizes both *incoming* cells (cells that affect the current cell—i.e., that appear in the current cell's formula) and *outgoing* cells (cells that are affected by the current cell—i.e., cells that have formulas that the current cell appears in). The system visually distinguishes the two kinds of cells by using color, line thickness, or other graphical characteristics. Our implementation groups incoming cells into spatially adjacent regions and encloses each region with a rectangle. The rectangle is connected to the current cell with a hairline, while outgoing cells are filled with gray. Figure 2 shows an example of a transient local view of a cell. Grouping incoming cells makes use of both the *logical* structure of the spreadsheet (they are inputs to the current cell) and the *physical* structure of the spreadsheet (they are adjacent in space).

The reason we call this technique *transient* lies in the manner in which the user specifies the current cell. In conventional spreadsheet applications, the user must move the cursor to a cell of interest and click in it to see the formula. In our system, the user specifies the current cell simply by moving the mouse cursor over the cell. When the cursor comes into a cell, the dataflow graph associated with the cell (incoming and outgoing cells) gradually appears on the screen (fades in), and it gradually disappears when the cursor moves away from the cell (fades out). Thus, the user can explore the dataflow graph structure of a spreadsheet simply by moving the cursor around the spreadsheet (see Figure 3).

This "mouseover" style of interaction is present in other systems. A notable example is Microsoft's "ToolTips," which are tiny windows containing a few words of

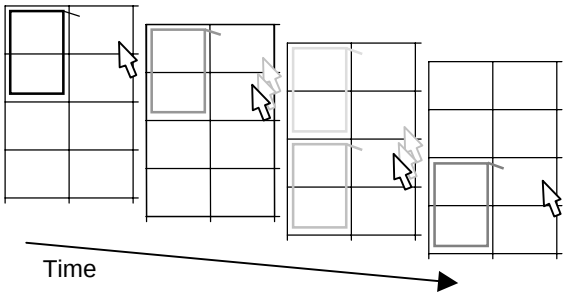

Figure 3. *Lightweight access to dataflow graphs*. The user can explore the dataflow graph simply by moving the cursor around. The graphs fade in and fade out for minimal visual disturbance.

explanatory text. ToolTips appear when the cursor lingers over an item. However, ToolTips are separate from the annotated objects and overlay them, while our presentation of the dataflow graph is integrated with the original view of the spreadsheet, not obscuring the spreadsheet and not becoming too obtrusive. The fading effect plays an important role in achieving this integration—it can smoothly present larger visual structures as the user moves the cursor about the spreadsheet. Without the fading animation and careful timing, constant flashing of the dataflow information would occur. In particular, when the dataflow graph is extensive, it would be very distracting to see large graphs appear and disappear frequently.

This transient interaction mode is suspended during some operations, like editing, so that incidental movements of the cursor will not cause unwanted visualizations.

4 Static global view

The transient local view has several advantages. It does not require special operations, so the user's focus can remain on the spreadsheet content as he simply moves the cursor around to invoke the local visualizations. It does not clutter the screen because it limits the presentation of the dataflow graph to that associated with one cell. Finally, the fading transition effect helps the user to view the global structure of a spreadsheet in a pleasing, non-jarring manner.

However, there are also many cases when the user may want to see the entire structure at once. For example, a user may want to quickly review the entire structure when he is handed a spreadsheet created by another user. We implemented the **static global view** to show this kind of overall structural information. In this mode, every dataflow connection appears on the screen—this means every incoming and outgoing cell for every cell in the spreadsheet (Figure 4). As in the transient local views, cells are presented as groups, which serves both to minimize clutter and to summarize the dataflow.

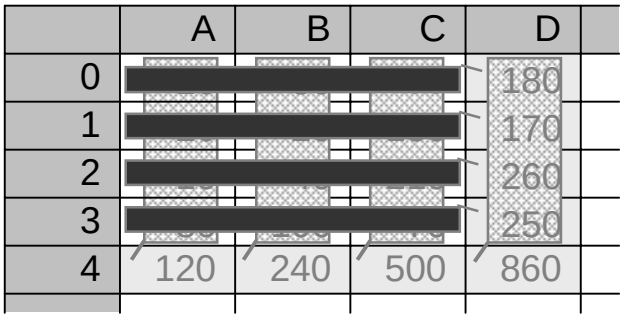

Figure 4. *Static global view*. The user can see the entire dataflow graph at once. The system uses different colors for different shapes of regions.

Even with the grouping technique, overlapping of many portions of the graph is unavoidable, and it can be difficult to follow the detailed dataflow for particular cells. Instead, the static global view is useful for getting a general overview of the entire structure. In our implementation, we assign different colors for vertical, horizontal and single-cell group structures. As a result, the user can quickly see how the spreadsheet is structured. This global view is also valuable in editing the dataflow structure (see Section 7.1).

5 Animated global explanation

The static global view works well to visualize the overall structure of the spreadsheet, especially when the dataflow graph has a regular pattern. However, for spreadsheets in which each cell is involved in many formulas, or in which the graph overlaps tightly in visually confusing ways, the static global view may not be very useful. In addition, the static global view does not effectively convey the general *direction* of dataflow.

To address these issues, we use animation to construct an unfolding *story* of the structure of the spreadsheet. Flows of cell values start at initial cells (that contain data only), join and split at intermediate cells, and end at terminal cells (typically, the result of a computation). We call this sequence the *narrative expression* of the dataflow graph, and the **animated global explanation** presents the story as a series of animations. It first examines the formulas to analyze cell dependencies, then determines an ordering for the graph, and finally shows the dataflow as a series of animations in which nodes at the beginning of the dataflow animate first, cascading into other animations as successive cells are visited. As a result, the user is presented with a visual demonstration of the march of computation within the graph. Figure 5 shows an example of the animated presentation.

In choosing the animation order of the explanation, we use information from both the underlying dataflow graph

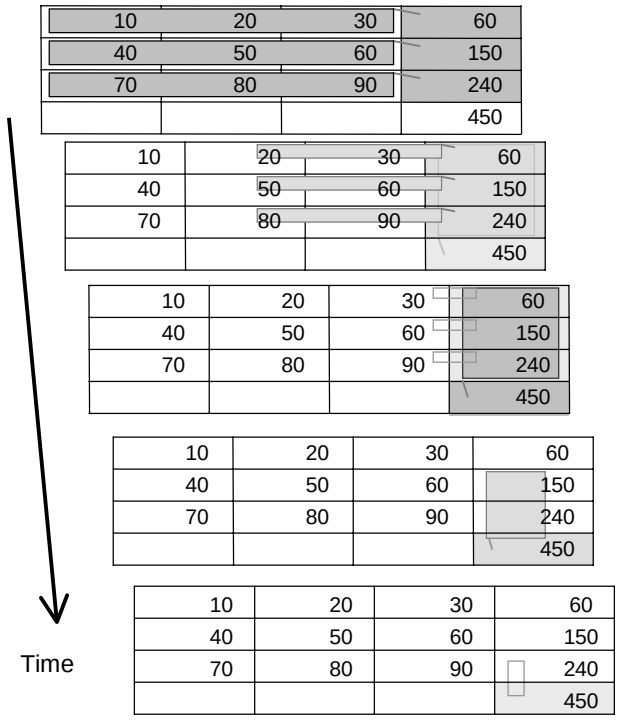

Figure 5. *Animated presentation of the dataflow structure*. The system automatically generates an animated global explanation of the dataflow structure of a spreadsheet. Incoming cells flow into their outgoing cells. We use fading effects throughout to create a smooth and pleasing animation.

and the tabular layout structure. Cell dependencies provide the basic ordering for the animation, but there can be many possibilities for simultaneous animation of various parts of the graph. Therefore, we also use information about the graphical layout of the spreadsheet to decide which animations to fire simultaneously. Just as the transient local and static global views use spatial relationships to group cells to form a more effective visualization, the animated global explanation uses spatial relationships to group *animations* to perform a more effective visualization. The goal is to produce an overall animation that sensibly organizes the dataflow into understandable chunks. In Figure 5, the animations of the first three horizontal flows occur simultaneously because these flows have regular spatial structure. Our current algorithm searches for these kinds of parallel dataflows to optimize into animation groups.

6 Semantic navigation

Excel's auditing function allows the user to move to spatially disjoint, but logically connected, cells by double-

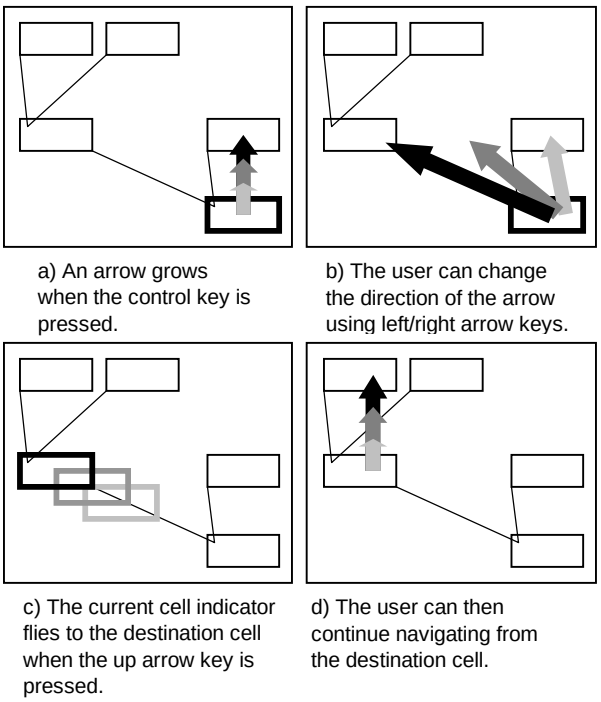

Figure 6. *Semantic navigation of dataflow graphs.* The user can navigate through the dataflow graph using arrow keys by holding down a control key.

clicking the arrowheads, a technique we call *semantic navigation.* In semantic navigation, the navigation cues are the semantic relations among cells rather than the superficial spatial continuity (Figure 6). We improved this semantic navigation interface to achieve easier access to the hidden structure by introducing keyboard-based operations and expressive animations.

We chose to make our semantic navigation available from the keyboard because it can streamline the "edit cell/ navigate to new cell" cycle. The current implementation works as follows:

1. The user begins semantic navigation by pressing the control key.
2. A large arrow sprouts from the current cell, growing to point to one of the semantically connected cells (incoming or outgoing cells). We call the cell that the arrow is pointing to a *destination cell* (Figure 6a).
3. The user can select the destination cell among the semantically connected cells using the left and right arrow keys (Figure 6b).
4. The user can jump to the destination cell by pressing the up arrow key (Figure 6c). The user can then continue to navigate through the dataflow graph by repeating 3 and 4 (Figure 6d).
5. When the control key is released, the cursor keys once again function for normal spatial navigation.

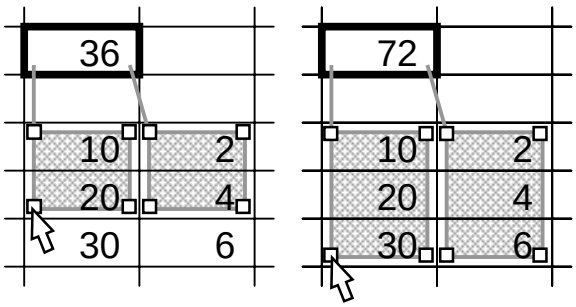

Figure 7. *Direct editing of dataflow graphs.* The user can transform the dataflow graph in ways similar to editing in object-oriented drawing editors. In this example, the user grabs a handle and drags it to scale two parts of the graph.

Every visual effect is presented in an animated manner to aid the user's comprehension of the navigation. The arrowhead moves continuously and the current cell indicator flies smoothly. This semantic navigation mechanism not only reduces the number of keystrokes, but also promotes better understanding of the dataflow structure. Easy access to logically connected cells makes the user feel that the cells are close to each other, even if they are spatially distant. In other words, semantic navigation visualizes the hidden topology of the dataflow graph through its characteristic interaction.

7 Visual editing

In previous sections, we discussed techniques to visually present the underlying dataflow structure of a spreadsheet. In this section, we will introduce techniques to visually *edit* the dataflow structure underlying the textual formulas. This visual editing functionality works surprisingly well in constructing dataflow structures that are spatially regular and simple, but complicated to specify textually. **Direct editing of dataflow graphs** allows the user to edit the dataflow structure in a way similar to object-oriented drawing editors, and **interactive graphical induction** enables the user to construct regular formulas for a series of cells interactively.

7.1 Direct editing of dataflow graphs

Direct editing of dataflow graphs is a straightforward application of dataflow graph visualization. It allows the user to move, scale, and delete the visualized dataflow graphs using standard direct manipulation techniques [27]. These editing operations cause the textual formulas to be updated accordingly. It is especially useful to edit multiple dataflow graphs directly at a time. (Excel allows graphical editing, but only of one region at a time.)

When the static global view is in effect or the user enters editing mode by clicking a cell in the transient visualization mode, dataflow graphs are made visible. The user can grab a graph by clicking it, move the selected constraint by dragging it, and scale it by dragging the handles that appear at its corners. Multiple graphs can be activated by clicking them with shift key pressed, causing all active graphs to move and scale simultaneously (Figure 7).

7.2 Interactive graphical induction

The visual representation of dataflow graphs makes it possible to understand the structure as a collection of regular patterns such as those shown in Figure 8. This section describes our approach of constructing these regular structures graphically using induction. In current spreadsheets, the user must use the "$" symbol to specify absolute references, and then correctly perform fill operations to construct the desired dataflow patterns. The $ symbol indicates that the following parameter is invariant throughout the fill operation, while other parameters change according to the relative position of the filled cells. For example, if the formula for B2 is `=$A2*B$1` and the user filled the region of B2-D4 with this formula, the formula for D4 becomes `=$A4*D$1`.

This scheme fails for a number of useful patterns, such as those shown in Figure 8. Even in simple cases it is very difficult to construct and understand correct formulas using $ symbols. Hendry proposed a programming-by-example technique to solve the problem [11]. His approach is to have the user key in the first two formulas, specify the region for filling, and run a menu command to initiate the fill. He showed that two examples are sufficient to express common spreadsheet structures. Formulas for the filled cells can be automatically inferred from the relative position of the cells and the difference of parameter values between the two examples. Although he doesn't use this term, he has essentially proposed filling formulas using *induction* on the structure of the first two formulas.

Our **interactive graphical induction** technique extends his approach by using graphical examples [7] instead of textual formulas for the induction. The system allows the user to edit the two examples graphically, and then visually presents the result of the fill operation. Our current implementation works as follows:

1. Key in a formula in the cell that starts the fill region.
2. Select the fill region and run the fill command.
3. The system presents a graphical proposal of the second step of the induction, which can be graphically manipulated into the correct state.
4. Press the confirm button to perform the fill operation.

The key idea is in step 3: the system prompts the user

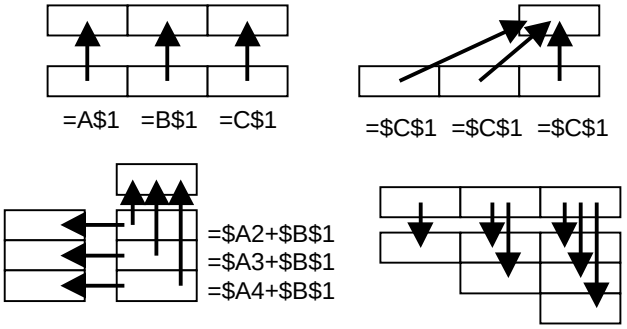

Figure 8. *Regular patterns of cell referencing.* It is difficult to encode these patterns using $ symbols. In fact, the fourth example cannot be expressed using $ symbols.

to specify the second example graphically. As a result of this interactive approach, the user no longer needs to prepare the textual formulas carefully before the fill operation. All the user has to do is to answer the system's request for confirmation, by very naturally moving the graphical boxes to the correct locations. In addition, the graphical representation of the first two examples and subsequent filled cells also helps the understanding of the resulting structure of the dataflow graph. Figure 9 shows examples of dataflow structures created using this graphical

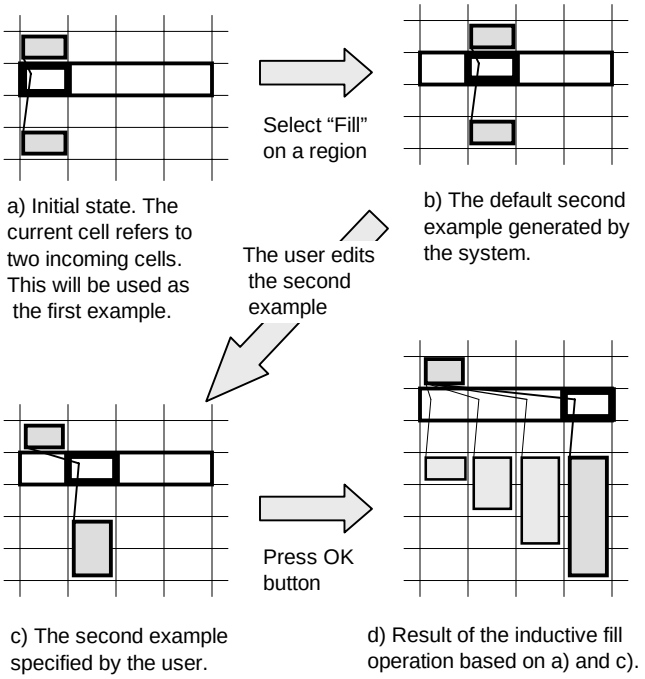

Figure 9. *Interactive graphical induction.* The system automatically infers the cell referencing for the filled cells based on induction from the first two graphical examples. The first example is given by the user. The system generates the default second example and asks the user to adjust it.

induction mechanism. Both position and size of regions are correctly inferred because the system applies induction for the row and column numbers of both the upper-left and lower-right cells of the fill regions.

8 Related work

The success of the spreadsheet as an easy-to-use computational environment [8, 12, 22] has led to several end-user programming systems based on spreadsheet-like interfaces. ACE [13] extended the basic idea of spreadsheets, a tabular layout enhanced by textual notations, and applied it to the development of interactive graphical applications. NoPumpG [29] applied the idea of the spreadsheet's automatic maintenance of predefined relationships among cell values to the control of graphical representations. C32 [19] used a spreadsheet interface for constructing constraints in user interface toolkits. Forms/3 [3, 10] is a general-purpose visual programming language based on the spreadsheet paradigm. Yang et al. proposed design benchmarks for visual programming [30] including static dataflow visualization.

Toolglass and Magic Lenses [2] use spatially dedicated lenses to visually present and interact with the underlying information layer. In contrast to a Magic Lens visualization of the dataflow structure of spreadsheets, fluid visualization controls the entire space in a coordinated way, providing a more integrated interface.

Zooming interfaces [1, 24] and distortion-oriented focus+context techniques [9, 15, 16] are efforts to present vast information spaces within a limited screen space. They try to present a focal part of the information space clearly while maintaining the surrounding context. For example, work on visualization of spreadsheets [26, 28] use focus+context techniques to visualize very large tabular layouts. The difference is that our focus is on the visualization of the hidden dataflow structure *behind* the tabular layout, and not that of the tabular layout itself.

Many modern user interface systems incorporate animation [3, 21]. Animations can give the feeling of solidity, continuity, and real existence to visual objects. They help the user to understand visual events by softening abrupt changes on the screen. They are indispensable tools to present a sequence of events in an intuitive way (i.e. algorithm animation [25]). In this paper, we proposed another useful application of animation effects: animations to reify and visualize invisible information. It is difficult to present hidden dataflow structures with static representations. Animation makes it possible to visualize these invisible structures.

Our graphical induction technique can be seen as an application of programming by demonstration/example techniques [7]. Metamouse [17] detects repetition in the user's graphical editing operations and suggests the next operation. Eager [6] analyzes a sequence of HyperCard operations and suggests the next set of operations. Chimera [14] and IMAGE [19] find appropriate geometric constraints from given graphical examples.

9 Discussion and conclusions

Spreadsheets have hidden dataflow graphs in addition to their superficial tabular layouts. We have presented a series of techniques that make the dataflow structure visible and accessible, while maintaining the original appearance of the spreadsheet. The goal of these techniques is to impart a better understanding of the dataflow structure by letting the user *visually* interact with these hidden structures.

We implemented these interaction techniques in our prototype spreadsheet program, and they noticeably improve the understandability of spreadsheets. Keys to our approach are carefully designed graphical integration of the dataflow information with the spreadsheet layout, animation of all graphical changes, and lightweight interaction. We found fading animations to be particularly effective—the user becomes subtly aware of the existence of the hidden dataflow graph at the position where it has faded away.

Future work includes integration with more realistic spreadsheet programs. As we mentioned earlier, our prototype spreadsheet program is written using Pad++ [1, 23] and Python running on Unix platforms. Although Pad++ provides powerful scaling and animation primitives for exploring novel ideas rapidly, it exacts a performance toll. As a result, smooth animation is possible only for spreadsheets containing 400 cells or fewer. Scaling our experiences to larger spreadsheets raises three issues: animation performance, algorithm performance, and visualization and navigation of large spreadsheet structures. We are confident that a custom implementation would support animation of larger spreadsheets comfortably. Our algorithms for supporting our various techniques rely upon traditional dataflow computations that are computationally tractable for larger spreadsheets. Visualization and navigation of very large spreadsheets, where cells refer to distant cells outside of the boundaries of the screen, is a more substantive issue. We expect that animated presentation and semantic navigation techniques would be especially useful for such spreadsheets. In these cases, automatic camera controls may be required, such as zooming out to provide a larger context, panning to a logically-connected cell, and then zooming in to permit a closer view.

Fluid visualization for spreadsheets is a particular application of our general set of fluid user interface techniques, whose goal is to provide lightweight, contextual, and animated access to a secondary layer of content while maintaining the appearance of the primary material. Fluid techniques allow the user to fluidly shift attention from primary to secondary content, as the system

fluidly alters its display to show previously hidden secondary content in the context of its associated primary content. Other domains in which we have explored fluid user interfaces include hypertext linking [31] and annotation [4]. Among possible further applications of the fluid visualization techniques are maps, CAD diagrams, and other graphical domains with rich underlying structures. Our experiences so far suggest that fluid techniques can be a powerful aid to users in a wide variety of situations that require comprehending and interacting with data.

References

1. Bederson, B.B., Hollan, J., Perlin, K., Meyer, J., Bacon, D., and Furnas, G., Pad++: a zoomable sketchpad for exploring alternate interface physics, *Journal of Visual Languages in Computing*, Vol.7, No.3, pp. 3-31, 1996.
2. Bier, E.A., Stone, M.C., Pier, K., Buxton, W., DeRose, T.D., Toolglass and Magic Lenses: The see-through interface, in *Proceedings of SIGGRAPH'93*, 1993.
3. Burnett, M., Ambler, A., Interactive visual data abstraction in a declarative visual programming language, *Journal of Visual Languages and Computing* 5(1), pp.29-60, 1994.
4. Chang, B.W., Mackinlay, J., Zellweger, P., Igarashi, T., A negotiation architecture for fluid documents, in *Proceedings of UIST'98*, to appear, 1998.
5. Chang, B.W., Ungar, D., Animation: from cartoons to the user interface, in *Proceedings of UIST'93*, pp.45-55, 1993.
6. Cypher, A., Eager: programming repetitive tasks by example, in *Proceedings of CHI '91*, pp. 33 – 39, 1991.
7. Cypher, A., ed., Watch What I Do: Programming by Demonstration, The MIT Press, 1993.
8. Fischer, G., Beyond human computer interaction: designing useful and usable computational environments, in *Proceedings of the HCI'93 Conference on People and Computers VIII*, pp.17-31, 1993.
9. Furnas, G.W., Effective view navigation, in *Proceedings of CHI'97*, pp.367-374, 1997.
10. Gottfried, H.J, Burnett, M., Graphical definitions: making spreadsheets visual through direct manipulation and gestures, in *Proceedings of IEEE Symposium on Visual Languages'97*, 1997.
11. Hendry, D.G., Display-based problems in spreadsheets: a critical incident and a design remedy, in *Proceedings of IEEE Visual Languages'95*, pp.284-290, 1995.
12. Hendry, D.G., Green, T.R.G., Creating, comprehending, and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model, *International Journal of Human-Computer Studies*, Vol.40, pp. 1033-1065, 1994.
13. Johnson, J.A., Nardi, B., Zarmer, C.L., Miller, J.R., Ace: building interactive graphical applications, *Communications of the ACM*, Vol.36, No.4, pp.41-55, 1993.
14. Kurlander, D., Feiner, S., Inferring constraints from multiple snapshots, *ACM Transactions on Graphics*, Vol.12, No.4 , pp. 277-304, 1993.
15. Lamping, J., Rao, R., Pirolli, P., A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, in *Proceedings of CHI'95*, pp.401-408, 1995.
16. Mackinlay, J.D., Robertson, G.G. and Card, S.K., The Perspective Wall: detail and context smoothly integrated, in *Proceedings of CHI'91*, pp. 173-179, 1991.
17. Maulsby, D., Kittlitz, K. and Witten, I., Metamouse: specifying graphical procedures by example, in *Proceedings of SIGGRAPH '89*, Vol. 23, No. 3, pp. 127-136, 1989.
18. Microsoft, Microsoft® Excel 97, http://www.microsoft.com/excel/.
19. Miyashita, K., Matsuoka, S., Takahashi, S., Yonezawa, A., Interactive generation of graphical user interfaces by multiple visual examples, in *Proceedings of UIST'94*, pp.85-94, 1994.
20. Myers, B.A, Graphical techniques in a spreadsheet for specifying user interfaces, in *Proceedings of CHI'91*, pp. 243-249, 1991.
21. Myers, B.A., Miller, R.C., McDaniel, R., Ferrency, A., Easily adding animations to interfaces using constraints, in *Proceedings of UIST'96*, pp.119-128, 1993.
22. Nardi, B., Miller, J.R., The spreadsheet interface: a basis for end user programming, in *Proceedings of IFIP INTERACT'90*, pp. 977-983, 1990.
23. Pad++ Reference Manual, http://www.cs.unm.edu/pad++/.
24. Perlin, K., Fox, D., Pad - an alternate approach to the computer interface, in *Proceedings of SIGGRAPH'93*, pp. 57-64, 1993.
25. Price, B.A., Baecker, R.M., and Small, I.S., A principled taxonomy of software visualization, *Journal of Visual Languages and Computing*, Vol.4, No.3, pp.211-266, 1993.
26. Rao, R., Card, S., The Table Lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information, in *Proceedings of CHI'94*, pp. 318-322, 1994.
27. Shneiderman, B., Direct manipulation: a step beyond programming languages. *IEEE Computer*, Vol.16, No.8, pp.57-69, 1983.
28. Spenke, M., Beilken, C., Berlarge, T., FOCUS: the interactive table for product comparison and selection, in *Proceedings of UIST'96*, pp.41-50, 1996.
29. Wilde, N., Lewis, C., Spreadsheet-based interactive graphics: from prototype to tool, in *Proceedings of CHI'90*, pp. 153-159, 1990.
30. Yang, S., Burnett, M., DeKoven, E., Zloof, M., Representation design benchmarks: a design-time aid for VPL navigable static representations, *Journal of Visual Languages and Computing* 8(5/6), Oct/Dec 1997.
31. Zellweger, P., Chang, B.W., Mackinlay, J., Fluid links for informed and incremental link transitions, in *Proceedings of Hypertext'98*, pp.50-57, 1998.