

# Side Views: Persistent, On-Demand Previews for Open-Ended Tasks

Michael Terry, Elizabeth D. Mynatt

Everyday Computing Lab, GVU Center  
College of Computing, Georgia Tech  
Atlanta, GA 30332-0280  
{mterry, mynatt}@cc.gatech.edu

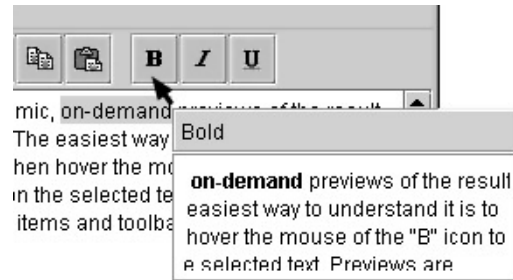
## ABSTRACT

We introduce Side Views, a user interface mechanism that provides on-demand, persistent, and dynamic previews of commands. Side Views are designed to explicitly support the practices and needs of expert users engaged in open-ended tasks. In this paper, we summarize results from field studies of expert users that motivated this work, then discuss the design of Side Views in detail. We show how Side Views' design affords their use as tools for clarifying, comparing, and contrasting commands; generating alternative visualizations; experimenting without modifying the original data (i.e., "what-if" tools); and as tools that support the serendipitous discovery of viable alternatives. We then convey lessons learned from implementing Side Views in two sample applications, a rich text editor and an image manipulation application. These contributions include a discussion of how to implement Side Views for commands with parameters, for commands that require direct user input (such as mouse strokes for a paint program), and for computationally-intensive commands.

## SUPPORTING OPEN-ENDED TASKS

We are interested in developing user interface mechanisms to support users engaged in open-ended tasks, which we define as those for which there exist no clear, predefined sequence of steps to arrive at an acceptable solution. Such tasks are unstructured, with goals that cannot always be precisely defined in advance. The nature of these tasks require people to approach them on a case-by-case basis, drawing upon past experience and knowledge to discover and construct an acceptable final solution [20]. Examples of open-ended tasks include traditional design problems such as those found in architecture, industrial design, and software design, as well as relatively smaller activities, such as writing a paper or color-balancing an image.

Solving an open-ended task requires domain-specific knowledge, but also requires more general-purpose problem-solving practices, such as performing experiments to discover the most viable approach to solving the problem. For example, when color balancing an image, an expert user may consider multiple operations to be equally applicable at



**Figure 1:** Side Views provide on-demand previews of the effects of commands within a pop-up window. In this figure, a preview of the "bold" command is shown applied to the current selection.

a particular moment. To assist with her decision, she may make copies of her image, then apply each operation to a different copy to find the best solution to her problem.

While many applications are designed to support *specific* open-ended tasks, there is a lack of user interface mechanisms crafted to support the broader, more general practices. For example, interfaces typically do not explicitly support the practice of experimenting with data (e.g., branching to try multiple alternatives in parallel). Instead, users must devise strategies such as saving back-up copies of their documents before experimenting, to ensure that they do not accidentally modify their original data beyond repair. However, requiring users to perform even such a small step can impose enough of a barrier to discourage these beneficial practices. The "undo" mechanism is a notable exception: Undo supports some aspects of open-ended tasks, such as simple experimentation, but it is relatively unique both in terms of its broad applicability and its versatility.

We would like to add to the interface designer's toolbox by contributing broadly-applicable user interface mechanisms geared towards supporting practices generally found in open-ended tasks. To this end, we present Side Views, a user interface mechanism that provides on-demand, persistent previews of a command (see Figure 1). Side Views use the tool-tip metaphor to present the previews: Side Views appear after a user hovers the cursor over an object in the interface for a short period of time. However, unlike a normal tool-tip, users can interact with the Side View, for example, choosing to view ranges of previews for

parameters (called parameter spectrums). Additionally, users can make Side Views persist, enabling comparisons between commands. These persistent previews can also be chained together to view the effect of multiple commands applied in succession.

In this paper we present the design of Side Views and demonstrate a range of uses for this tool, focusing on those that enhance practices commonly employed by *expert users*. We argue that Side Views offer a natural extension to an existing, lightweight user interface mechanism (tool-tips), while allowing users to:

- perform simultaneous comparisons of *multiple commands*;
- compare multiple instantiations of the *same command*, thus aiding in the efficient and informed selection of a command and its parameters;
- create *alternative visualizations* of their data;
- experiment with possibilities without committing to any changes to the original data (“*what-if*” tools);
- *customize* the interface to enhance workflow; and
- *serendipitously discover* viable alternatives to a planned course of action.

The rest of the paper is structured as follows: First, we provide a more detailed description of common expert practices and needs in open-ended tasks, derived from field interviews and observations of expert users, and prior research in this domain. We then contrast these user practices with those supported by current user interfaces, uncovering limitations in the process. The design of Side Views is then presented, and we show how Side Views address some of these limitations through their various uses.

Next, we convey lessons learned from implementing Side Views in two applications, a rich text editor and an image manipulation program. These lessons include contributions of interest to implementers and designers, such as how Side Views can be implemented for commands with parameters, for commands that require direct input (such as text input in a word processor, or pointer input for a paint program), and for computationally-intensive commands. We conclude with a review of related work, and opportunities for future research.

## CHARACTERIZING OPEN-ENDED TASKS

To understand typical expert practices in open-ended tasks, we conducted field interviews and observations of professional users of an image manipulation program. The users we observed employ this application in two primary tasks, image toning (e.g., color correction), and the design and creation of user interfaces. As we discuss these findings, it is important to keep in mind that these individuals are experts and fully understand their task and tools; the processes described are not those employed by novices unfamiliar with the interface or the problem space.

Our study revealed a common set of *intertwined* practices that echo those described by prior research investigating the design process (e.g., [19, 20]). These practices can be summarized as *experimentation*, *branching*, *evaluation*, and *iteration*.

*Experimentation* comprises discovering available options and testing out hypotheses. In a computer application, this process includes such activities as trying and undoing various commands to find the best option amongst a *set* of commands, as well as trying and undoing different parameter settings for the *same* command.

*Branching* is a process of exploring multiple paths in parallel, and is typically enacted through such strategies as saving duplicate copies of a document, or embedding multiple versions within the document itself (for example, creating alternative versions of a user interface widget side-by-side, or multiple versions of the same paragraph, one after the other).

*Evaluation* serves to assess one’s progress with respect to the envisioned solution and to past states. Evaluation can occur directly (e.g., placing different versions side-by-side) or indirectly (e.g., between the current state and an envisioned future state, or a remembered past state). Additionally, and importantly, evaluation can occur using *alternative* representations of the data. For example, in image toning an individual may assess one’s progress by examining the individual color channels of the image. Alternative representations help the user ignore some details of a problem while concentrating on others [19, 20].

*Iteration* is the process of incrementally solving portions of a problem, and building upon and combining these individual solutions.

As mentioned, these four basic processes are interconnected and are often used in concert with one another. For example, evaluation occurs whenever one experiments with several different commands, searching for the “best” one.

While we observed these practices in the context of computer-based activities, these practices are even more evident in paper-based design activities, given the flexibility of that medium (e.g., as found in [19]). However, as we discuss next, the current design of computational tools prevents users from fully engaging in these practices.

## Clarifying the Challenge for User Interface Design: Breaking from the Single State Document Model

While some user interface support does exist for open-ended tasks, it is typically limited, requiring users to develop workarounds to compensate for an interface’s deficiencies. For example, “undo” supports near-term experimentation, as it allows users to sequentially try and undo commands to discover the best operation. However, we also observed its use as an evaluation mechanism: In the image manipulation application, users often apply a command, then quickly undo and redo the command in rapid succession to effectively “flash” the current and previous versions on the

screen. This practice affords the comparison of these two versions, but in time, rather than side-by-side. Arguably, this use of “undo” veers away from its original intent, and is more indicative of a workaround intended to bend an interface to conform to a user’s desired practices.

### The Need for Multiple Realities

The need for such workarounds becomes clear if we contrast an expert’s desired practices with the model of interaction typically employed by current user interfaces. In particular, the desired practices usually involve multiple versions of the user’s data: Experimentation revolves around testing multiple hypotheses; branching, by definition, results in multiple versions; evaluation is the comparison of different states; and iteration builds upon prior work. Thus, as part of developing a solution to an open-ended task, a user makes use of multiple realities of the problem space and solution. However, current interfaces typically employ a model of interaction that recognizes only one document state at a time. We define this interaction model as the *Single State Document Model*.

### The Single State Document Model

The Single State Document Model is an interaction model that requires a document to be in one state, and one state only, at any point in time. Users progress through tasks by applying a single operation, then operating on the changed state.

Though conceptually simple for both user and implementor, this interaction model imposes a serial, linear progression through a task that is a poor fit to the practices of experts engaged in open-ended tasks. Thus, while a user may need to simultaneously consider multiple alternatives, the Single State Document Model creates a form of interface “tunnel vision” that limits the number and types of views and actions available at any one moment. As an example, the image manipulation program we studied offers six variants of “blur.” The only way to unambiguously differentiate between these different versions of blur is to actually try each in turn, since multiple previews are not simultaneously available. Similarly, in our field studies, we often observed that experts spend time experimenting with parameter settings to tweak a command’s effect. Though the experts could often intuit the area of most interest, they still needed

time to fine-tune and confirm their choice of settings since there is usually at most a single preview available for a command.

While the need to experiment, create multiple versions, iterate, etc., is strongly advocated in the actual practice of designing user interfaces, these same processes are seldom transferred into, and supported by, the interfaces that result from these processes. Prior work has recognized these limitations in existing interfaces [16], and argued for a “subjunctive interface,” or an interface that enables users to simultaneously explore multiple alternatives in parallel [17]. Side Views continues in the spirit of subjunctive interfaces by providing users with a lightweight mechanism to create multiple, simultaneous views of potential future states via persistent previews. We turn now to a detailed description of its design.

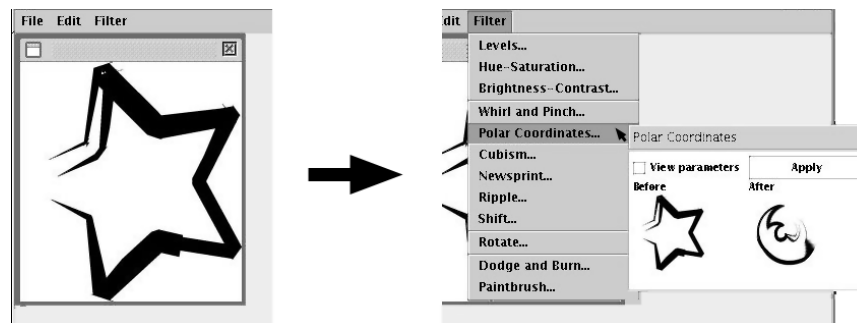
### SIDE VIEWS: DESIGN

Side Views employ the tool-tip metaphor to provide on-demand previews, but extend their basic design in five main ways. Side Views:

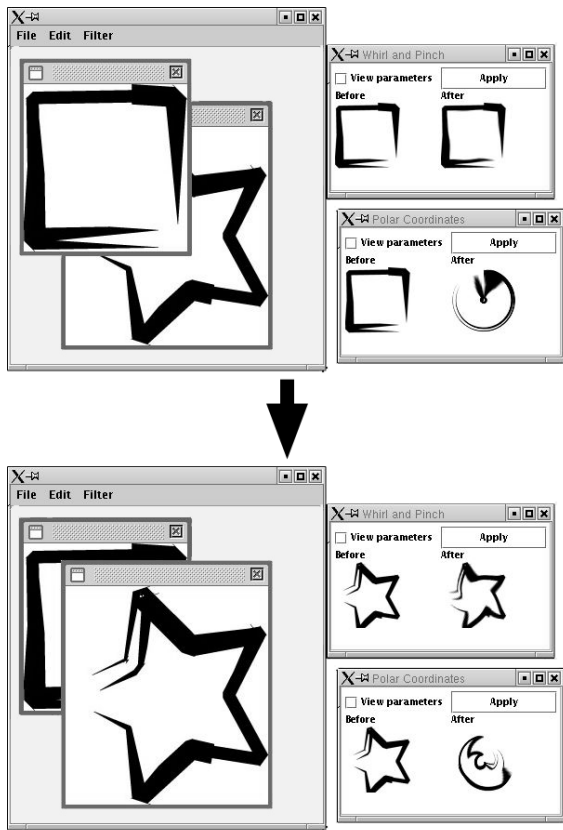
1. display *dynamic previews* in the pop-up window of a command or user interface object, as applied to the current context,
2. support *interaction*,
3. can be made to *persist*,
4. scale to commands with multiple parameters through *parameter spectrums*, or ranges of previews for a command, and
5. can be chained together to allow for *function composition*.

### Dynamic Previews

Side Views provide dynamic, on-demand previews of a command applied to a copy of the current data, similar to the previews often found in modal dialog boxes (see Figure 2). Side Views also provide previews of the effects of other user interface objects, such as sliders or radio buttons. For example, in a dialog box of font styles, hovering over the checkbox for the “underline” option elicits a Side View previewing the user’s selected text underlined.



**Figure 2:** A Side View for an image manipulation application. The left illustration shows the current document, the right a Side View for the “Polar Coordinates” filter that appears after the user hovers the cursor over the menu item.



**Figure 3:** Side Views automatically update their preview based on the active document, as shown above.

### Interaction

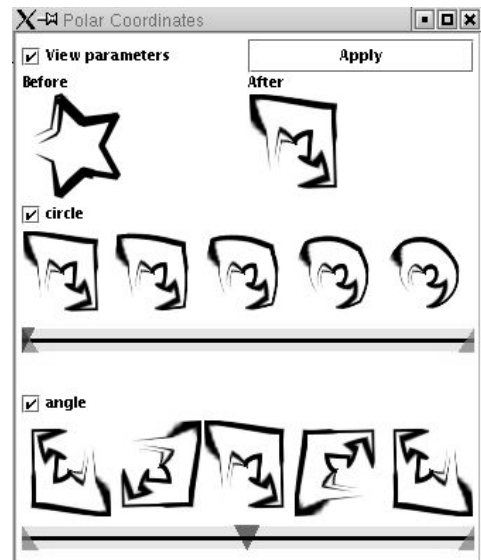
Users can interact with a Side View, for example, resizing it, varying its parameters, or invoking the command it represents. We discuss parameters and the issue of what content to preview more fully below.

### Persistent Side Views

When a single and/or fleeting preview is not enough for the user's needs, users can interact with a Side View to make it persistent (i.e., turn it into a regular window that does not automatically dismiss itself after a short interval). Persistent Side Views continually update their previews to reflect the current state of the active document. That is, as modifications are made to the active document, the selected region is changed, or the active document is switched to another document, a persistent Side View automatically updates its preview to reflect those changes (Figure 3).

### Parameter Spectrums

Side Views for commands with parameters initially show a single preview using the default (or the user's last) settings for a command. When a user desires to see and/or interact with a broader range of possibilities for a single command, Side Views can be expanded into *parameter spectrums* (Figure 4). Parameter spectrums show a series of previews across the range of values for each parameter. Initially, the range displayed is a sampling of all possible values for a



**Figure 4:** A Side View showing parameter spectrums for two of the filter's parameters. Note how each parameter varies its previews only on its own dimension.

parameter. However, the user can vary the range previewed to focus on a specific set of values.

Each spectrum varies its previews only on the parameter it represents. For example, if there are two parameters for an oval, height and width, the parameter spectrum for height uses the current setting of the width parameter, and varies its previews in the height dimension (Figure 4 illustrates this property in an actual filter). When one parameter's current value is changed, all other parameter spectrums update their previews to reflect this new value. This behavior enables a user to interactively vary one parameter's settings to understand how it affects the other parameters.

### Function Composition

To view previews of two or more Side Views combined (function composition), users can chain Side Views together. In our current implementation, chaining commands is supported by dragging and dropping one preview on the other. Performing this action causes a new Side View to appear that previews the effect of both commands at once.

As with Side Views for a single command, users can view and vary each command's parameters individually. Changing the parameters in one Side View causes subsequent Side Views to update their previews, since later Side Views are dependent on the output of former Side Views. The effect is one where users can view changes "ripple" down the chain of Side Views.

### Putting Side Views to Work

Side Views' feature set makes them amenable to a range of tasks. At a high level, Side Views enable users to more easily perform breadth- and depth-first searches of possibilities, without committing to any one course of action. In essence, Side Views foreshadow potential future

states so users can make more informed decisions, and spend less time hunting for suitable operations and parameter settings.

At a more concrete level, Side Views can be used to *clarify* a command's effect, to make *comparisons*, to create *alternative visualizations* of the user's data, to experiment ("what-if" tools), to *customize* an interface, and as a mechanism to *serendipitously discover* viable alternatives.

#### *Clarifying the Effect of a Command*

The most basic use of Side Views is to clarify the effect of a command. Previewing the effect of a command (or its range of values in a parameter spectrum) relieves users from the need to internalize a model of a command and the effect of its parameters to guide their searches. This aid is especially useful for parameters that have seemingly arbitrary ranges of numerical values. Thus, instead of forcing the user to conform to the interface, Side Views represent the output of a function in a way that can be more easily appropriated and used by a person.

#### *Supporting Comparisons Between Commands and Parameter Settings*

Persistent Side Views enable direct, side-by-side comparisons between commands (as shown in Figure 3). For example, several commands, may seem equally viable at a particular moment (e.g., the six variants of blur mentioned earlier). Persistent Side Views offer the user the opportunity to simultaneously view all desired options at once, delaying commitment to any one command until an informed decision can be made.

Users can also instantiate the same command multiple times to perform comparisons between completely different parameter settings for the same command. This feature is beneficial for complex commands that create a large space of choices to search.

#### *Side Views as Alternative Visualizations*

Side Views provide persistent, dynamic previews of commands, affording their use as *alternative visualizations* of the current data. When used in this fashion, Side Views' previews serve as "lenses" through which users can examine their data, similar in effect to Magic Lenses [8]. For example, if a user is painting an image that will be printed in both color and grayscale, a Side View for a grayscale-conversion command provides a grayscale visualization of the document that the user can refer to when choosing colors and content that will print well in both versions.

These alternative visualizations can also be used for evaluative purposes: the alternative representations can serve to emphasize certain features, while de-emphasizing others, depending on the data and the operations chosen.

#### *Side Views as "What-If" Tools*

Side Views' multiple views of potential future states make them useful "what-if" tools that let users experiment without requiring any changes to the original data, and without requiring the users to create duplicate copies of their data.

This ability can help foster exploration by lowering the barrier to simple experimentation.

#### *Supporting Customization*

Many tasks become routinized over time, yet still require thoughtful user input for each of the steps. For example, image toning is a process in which a user makes adjustments to the color balance of a digital image so that it will reproduce well in a target medium (such as a newspaper or web browser). Typically, a small set of commands forms the core set of operations a user will apply to an image, and the sequence of operations will be the same for a similar batch of images. However, each operation still requires the fine-tuning of parameters on the part of the user, eliminating the possibility of creating and using a macro or a script.

In such situations, users can instantiate Side Views for the most frequently used commands, and arrange them in convenient locations around the interface. For common sequences, users can chain several Side Views together. Returning to the image toning task, users can tone one image, then rely on Side Views' parameter spectrums to quickly find and modify settings for subsequent images.

#### *Facilitating Serendipitous Discovery*

Both persistent and transient Side Views can facilitate serendipitous discovery of desirable commands and parameter settings as one works. For example, as a user becomes mechanical and routinized in her interactions with an interface, there is less likelihood of the user "exploring" the interface to ensure her choice of command and parameters is the best. Side Views can facilitate the accidental discovery of viable alternatives to a planned course of action when they pop-up through normal interactions, or through the simultaneous use of multiple, persistent Side Views.

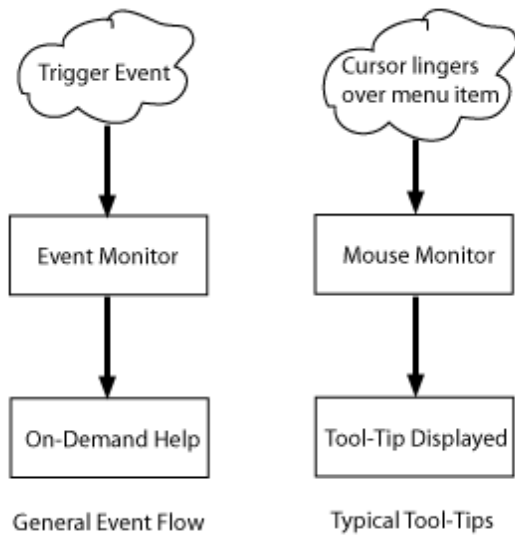
In sum, Side Views provide mechanisms that open up the interface, making it less like a "black box," while providing better support for expert practices such as experimentation and evaluation.

## **IMPLEMENTING SIDE VIEWS**

We have implemented Side Views in two applications, a rich text editor and an image manipulation program. Both were written in Java, with the GNU Image Manipulation Program (the GIMP) [12] used as the image manipulation engine for the latter application<sup>1</sup>. Using the GIMP grants us a production-quality, fully-featured image manipulation engine that enables us to test out Side Views in authentic situations.

---

1. To use the GIMP as the image manipulation engine, we wrote a general-purpose plug-in for the GIMP that enables Java code to issue commands to the GIMP, and read and write pixel data. This extension will be available for download from: <http://www.cc.gatech.edu/fce/ecl>



**Figure 5:** A general and specific flow-diagram for the instantiation of a tool-tip. Tool-tips monitor a trigger event (typically a cursor lingering over an object such as a menu item), then display on-demand help, such as a text-based tool-tip.

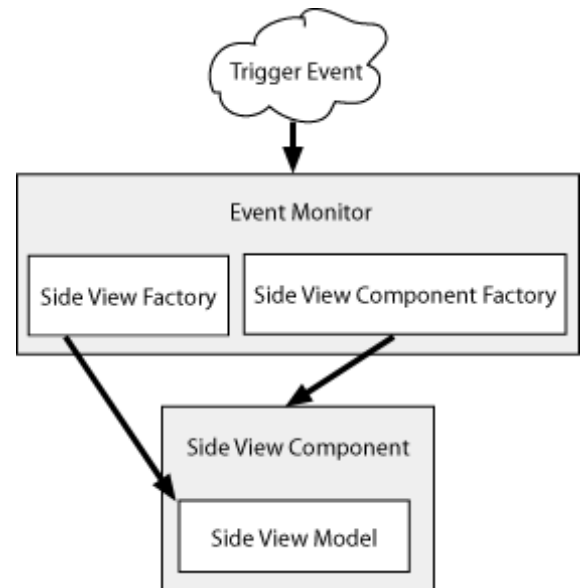
In this section, we describe the Side Views' architecture, their interaction semantics, and the lessons learned from building the two applications.

### Architecture

Side Views' architectural design is intended to be flexible and extensible. It is a modular, pluggable design that makes use of design patterns [11] to factor out behavior and functionality that may differ across applications and user needs. While our first implementation is within a GUI, the design itself is not explicitly tied to a GUI, and could be used within other interfaces, such as one whose primary mode of input is speech.

The design is driven by the observation that on-demand help is triggered by a specific event, but that this event and the form of help may vary (see Figure 5). For example, a traditional tool-tip is shown after the mouse hovers over an interface object for a short period of time. However, it is reasonable to assume that the trigger event could arise from keystrokes, and that the help given could take another form, such as synthetic speech, rather than visual cues alone. Thus, the design of Side Views breaks the architecture into components that monitor an interface for events, and factory classes to generate and activate application-specific Side Views when needed. Figure 6 shows an overview of this design.

Given this overview of the basic architectural design, we turn now to the specific interaction semantics of the Side Views we created, followed by implementation issues.



**Figure 6:** Side Views' basic architecture. One or more monitor objects "listen" for trigger events. When one is detected, two factory classes, one for Side Views, another for a component that activates and "renders" the Side View, are used to instantiate and display a Side View. In an MVC architecture, the Side View is the model; the Component, the view and controller. Applications can create custom monitors, Side Views, and Side View Components.

### Interaction Semantics

#### Invocation

The invocation of Side Views within our two sample applications is identical to that of normal tool-tips: hovering over a user interface object (such as a menu command) causes a Side View to appear after a delay. Existing tool-tips typically appear after a 750ms delay. However, in our informal tests users requested a shorter delay because they wanted faster access to Side Views when sampling the interface. Thus, our current implementation uses a much shorter 400ms delay. Our hypothesis is that the additional utility of Side Views for experts motivates the request for a shorter delay.

#### *Instantaneous Display of New Side Views after Initial Display*

As with normal tool-tips, if a transient Side View is visible, other Side Views will instantly appear if the cursor moves to a new object. This instantaneous appearance of new Side Views is important as it facilitates rapid sampling of options within the interface: After a Side View has been made visible, users can simply sweep the interface with their cursor, pausing over items of interest to discern their effect, rather than having to wait the default delay for each interface object.

#### *Making a Side View Persistent*

To make a Side View persistent, users click in the title bar of a Side View. This behavior changes the Side View into a "regular" window that does not automatically disappear.

Persistent Side Views dynamically update their previews to reflect the current state of the active document.

### *Extended and Suspended Dismiss Delays*

Normal tool-tips automatically hide themselves after a certain amount of time (~7 seconds), called a dismiss delay. However, Side Views offer a much more information-rich preview, and thus can require more time to view. Thus, we extended Side Views' dismiss delay to 10 seconds, but also added the ability for the delay to be temporarily halted. When a user's cursor enters a transient Side View, the dismiss timer is stopped until the cursor leaves the Side View.

### *Supporting Interaction*

Supporting interaction in a transient Side View requires a modification to typical tool-tip implementations, because normal tool-tips hide themselves whenever the user moves the cursor out of the component that generated the tool-tip. For transient Side Views to support interaction, we introduce a "grace period" whereby the Side View remains visible for a short period of time after the cursor leaves the triggering object. Currently, we use a 750ms delay for this value: after the cursor leaves the object that initiated the Side View, the Side View will remain visible for an additional 750ms to give the user time to enter a transient Side View. If the cursor enters another interface object (e.g., another menu item), then the original Side View is immediately hidden and a Side View for the second object is shown.

After the cursor enters the transient Side View, an identical 750ms delay is used before dismissing a Side View when the user's cursor leaves the Side View. This delay takes care of the case in which the user accidentally enters and exits a transient Side View when she actually intends to interact with it.

## **Implementation Issues**

### *Preview Content*

Determining what information to preview can be challenging for some commands. For commands that produce a visual change in the interface, one challenge is choosing how to render previews for changes that are too large for a Side View window. For example, if a user selects all the text in a multi-page document and wishes to preview a style change (such as a different font size), there is a question of how the Side View should render this rather large preview.

For commands for which there are no visual changes in the interface (such as the "print" icon in a toolbar, which automatically prints the current document using the last settings), the choice of what to preview is also unclear.

There are a number of approaches one could take to address these concerns. For example, for changes to large selections in a text document, the Side View could show the most recently edited text, the beginning or end of the current text selection, a scaled-down version, and so on. However, through the design, development, and use of Side Views, it

appears that no single heuristic can reliably predict the content of most interest to the user at any point in time -- there are simply too many special cases to consider. Therefore, in our implementations, we chose to apply a consistent, predictable algorithm to the initial display of each Side View, but allow the user to modify the preview. For Side Views in the text editor, we align the beginning of the text selection in the top-left corner of the Side View, unless the content is at the right margin, in which case we right-align the preview (as in Figure 1). In the image manipulation program, transient Side Views always appear at a fixed thumbnail-size that displays a "before" and "after" preview. These previews can later be resized to show larger previews, and users can choose to view the parameter spectrums as well. The initial preview uses the command's last settings.

For commands that do not produce a visual change, our rule of thumb is to present a summary of the effect of the command. For example, the Side View for the "print" icon could show a print preview, but this is largely unnecessary since most applications are "WYSIWYG." Instead, the Side View could display the printer settings that will be applied: the printer that will be used, the number of copies, the quality, etc.

### *Parameter Spectrums*

As described in the design section, parameter spectrums originally display a sampling of previews across the full range of possible values for each parameter. Users can vary the range shown in two ways. First, they can directly vary the boundaries of the range via a slider that has three handles. The center handle acts as a "normal" handle in a slider, selecting the current value. The outer handles serve to vary the boundaries of the range of values shown (see Figure 7).

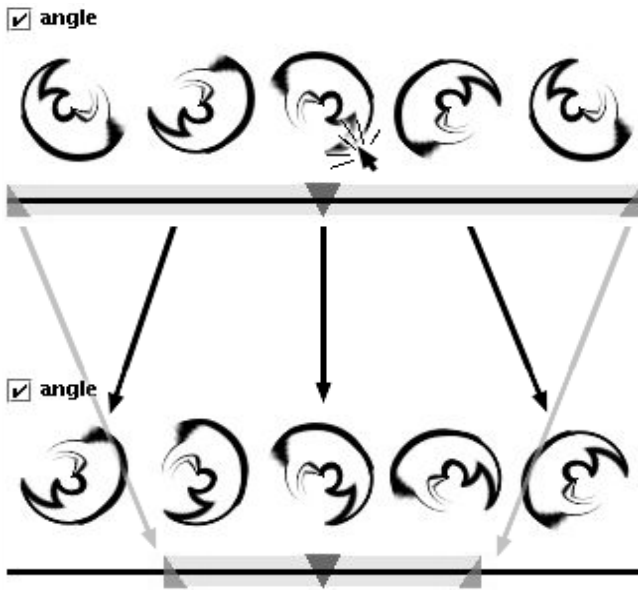
Second, users can click on a preview of interest. When a user clicks a preview, the values of the previews to the left and right of the one chosen become the new boundaries for the range of values shown (again, refer to Figure 7.)

### *Previewing Direct Input*

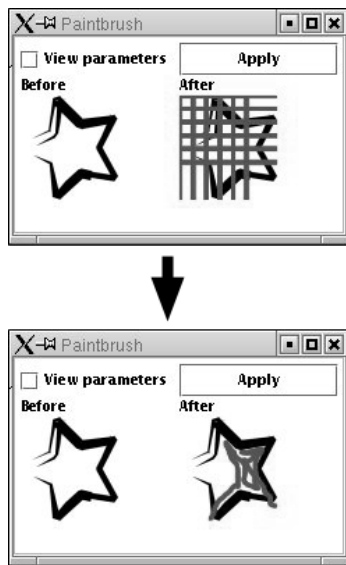
Many operations require direct input, such as text entry or mouse input, and some settings affect how that input is interpreted. For example, the size, color, and opacity of a paintbrush influence how a user's mouse strokes will be "painted" on a canvas.

Side Views for commands that require direct input present a unique challenge, in that the data to be previewed has not yet been entered. Yet it is desirable to be able to preview the settings for the tool before actually using the tool.

In these circumstances, Side Views mimic user input in their initial previews, then support user interaction to explicitly input data to preview. For example, the Side View for a paintbrush initially shows an overlapping grid of strokes that enables the user to see the effects of both single and overlapping paint strokes on their image. If the grid does not provide the information needed, the user can move the



**Figure 7:** Sequence of a user narrowing the range of values previewed in a parameter spectrum. In the top figure, the user clicks on the preview in the center, signaling his interest in that value. The interface responds by pushing neighbors on the left and right of the chosen preview to the ends of the spectrum (the black arrows). At the same time, the slider updates the positions of the handles for the lower and upper bounds, to indicate the new range of values previewed (the gray arrows)



**Figure 8:** Previews for commands requiring direct input initially mimic user input, but also allow explicit user input. The top illustration shows the default grid of paint strokes for the paintbrush Side View. The lower Side View shows the grid cleared away and replaced by a user's mouse strokes over the original image.

cursor over the original image. This action clears the default grid in the Side View and replaces it with the user's current mouse strokes (see Figure 8). These strokes persist after the cursor has left the image, allowing the user to interactively

vary parameters and view the effect in the sample paint stroke drawn. This capability effectively enables users to *retroactively* vary the parameters for previews of commands with direct input.

### Computationally Expensive Commands

Not all commands can be instantly previewed. Generating a preview might be computationally costly, by virtue of the operations performed, the amount of data that needs to be copied, etc.

In our implementations, we have addressed this problem in several ways. First, we create threads that perform background rendering. These threads enable the interface to remain responsive, while providing updated previews as they become available. Second, in the image manipulation application, we first scale an image down to its thumbnail size, then apply the required filter. This procedure quickly produces accurate previews for most operations (such as those that modify an image's colors, its rotation, etc.), but not all -- some filters are not commutative with respect to scaling transformations. That is, a filter will produce different results if the image is first scaled then transformed, rather than vice versa. Thus, another background rendering thread is required to apply these filters to a full-scale version of the image before scaling. Because we have been working primarily with operations that are commutative with respect to scaling transformations, we have only implemented the first tier of rendering threads at this time. We plan to add the second set of rendering threads for those commands that require them; an open research issue is how to decorate these previews so that users understand that a more accurate preview is on its way.

### RELATED WORK

Many applications extend tool-tips' basic functionality beyond static, text-based help. For example, a number of Corel products [10] (e.g., WordPerfect and CorelDraw) offer stationary tool-tips for font selection that dynamically preview font attributes using the current document's text. Programming tools are also increasingly equipped with on-demand help features, such as tool-tips that show the current value of a variable within a debugger, or source code editors that can display a list of applicable methods for a given object (see VisualAge for Java [13] for examples of both features). However, we know of no existing form of on-demand help that provides the breadth of features offered by Side Views, such as the capability to view multiple, dynamically-generated previews.

Work in subjunctive interfaces [16,17] has developed mechanisms that allow parallel document states to coexist simultaneously. The primary method used to represent these multiple realities is a layering of the (partially transparent) states on top of each other in the document window. Side Views also enables multiple potential states to be viewed simultaneously, but relies on windows separate from the main document window to show the previews. This design choice helps avoid cluttering the main document window, and is also essential in image manipulation tasks, for which the layering of the multiple states would not be feasible.

Side Views address similar problem areas as See-Through Tools [7] (i.e., Toolglasses and Magic Lenses [7,8,21]). Side Views both complement and extend this previous work. See-Through Tools employ the metaphor of a physical lens that transforms the underlying data representation via a filter. Users can “click-through” the tool to apply the command previewed. Like Side Views, these tools unambiguously present available options to the user, allow alternative visualizations of the data, and let the user explore options without actually modifying the data. Side Views complements See-Through Tools by introducing a mechanism through which these types of tools can be easily instantiated within current user interfaces, using existing interface metaphors.

Side Views differ from See-Through Tools in that Side Views do not use a spatial relationship to define the source of the data previewed, but rather are tied to the currently active data. This design decision enables mechanisms such as parameter spectrums, and also allows for more flexible layout of the tools. For example, a user can place Side Views in available, peripheral portions of their display (e.g., on a second monitor) instead of in their primary workspace. This feature is essential for certain tasks such as graphic design or image manipulation in which the user must be able to constantly view the document by itself for evaluation purposes. The spatial decoupling of the tool from the data also enables Side Views to provide multiple, simultaneous previews of the same data using different commands. Users can thus perform side-by-side comparisons of alternative options.

Fluid links and fluid documents [9, 22] offer techniques for providing additional information on-demand, for example when a user hovers the cursor over a link. Fluid documents display additional information in the context of the document itself, rather than in a separate pop-up window. Side Views share some goals with Fluid documents, though Side Views focus on the ability to generate multiple, simultaneous previews of commands.

Chateau, a suggestive interface for 3D drawing [14], uses current selections in a 3D model to create an array of thumbnail previews showing potential future states based on those selections. Like Side Views, Chateau provides a user with an array of previews to help guide her search. Side Views differ in that they are user-initiated, giving the user more control over which previews to show, and when they should be shown.

Previous work has attempted to make command icons (e.g., in toolbars or palettes) more readily understood by animating their functionality when the cursor enters the icon [3]. Like Side Views, this technique aids novices, but its use of canned animations limits its utility once an interface has been learned.

Our implementation of parameter spectrums contains some interaction semantics reminiscent of those found in the Alphaslider [2]. For example, both show ranges of values,

and both can narrow the focus on the range of values shown. However, Side Views’ parameter spectrums provide explicit handles to allow the user to manually vary the range of values previewed.

Adobe Photoshop [1] includes a “Variations” command that generates a ring of previews representing different color balance operations. Clicking on a preview applies that operation within the dialog box, and all previews update to reflect the new setting. This tool echoes Side Views’ parameter spectrums, but parameter spectrums improve upon Variations in several ways. First, parameter spectrums show ranges of values per parameter, granting the user a better sense of the *range* of possibilities for each parameter. Second, navigation within parameter spectrums is much more powerful since users can directly access and manipulate values and the ranges previewed.

When Side Views are chained together, the user can view how changes in parameter settings in one command affect subsequent Side Views. This functionality echoes that of Editable Graphical Histories [15]. Editable Graphical Histories enable a user to view and modify past operations, and see the results trickle down the chain of operations to the present state. Combining these techniques would be beneficial, as it would create a system in which the user can access and modify both past and future points on an “interaction timeline.”

Design Galleries [18] is a system that automatically samples a function’s parameter space to identify perceptually distinct output within that space. While parameter spectrums show interpolations across a function’s parameters, Design Galleries attempt to inform a user’s search by first finding and displaying visually distinct elements. Because Design Galleries does not specifically address interaction and the manipulation of parameters once a likely candidate has been identified by the user, the two tools may nicely complement one another.

## SUMMARY AND FUTURE WORK

We have presented the design and implementation of Side Views, an interface mechanism to support expert users in open-ended tasks. Side Views take advantage of increasingly powerful computers to generate multiple views into potential future states, a task that is otherwise tedious and time-consuming for users, yet of great benefit.

Our next steps are to perform user testing, and to integrate Side Views into other applications. As a quick test of the latter, we integrated the University of Maryland’s PhotoMesa [4] into our image manipulation application. The combined software enables users to browse images using PhotoMesa, while simultaneously seeing the currently selected image through any visible Side View filters: When the cursor hovers over an image in PhotoMesa, PhotoMesa’s default behavior automatically shows a tool-tip consisting of a larger thumbnail of that image; in our integration, all Side Views update their previews to use that larger thumbnail’s image as their source. This integration has proven extremely

useful for browsing images that require some manipulation for proper viewing. For example, if there are images that need to be rotated to be properly viewed, a Side View for the rotation command can be instantiated and referred to while browsing the images in PhotoMesa. This browsing process is fluid and fast, and does not require users to manually open images nor apply a command to see the corrected images.

What is especially promising about combining these mechanisms (i.e., ZUI's [5,6] and Side Views) is the potential for creating an environment in which the user can more easily work with multiple documents *and* multiple potential future states. Just as importantly, this combination of ZUI's and Side Views suggests a method to manage the visualization and navigation of a large number previews simultaneously, something that the current implementation of Side Views does not cleanly handle.

Finally, there remain many interesting research challenges for the integration of Side Views in other application domains, such as those concerned with multimedia. For example, we are interested in understanding how the concepts might be applied to audio and video editing.

## REFERENCES

1. Adobe Systems Incorporated. <http://www.adobe.com>
2. Ahlberg, C., & Schneiderman, B. The Alphaslider: A Compact and Rapid Selector. In *Proceedings of CHI '94*, pp. 365-371.
3. Baecker, R., Small, I., Mander, R. Bringing Icons to Life. In *Proceedings of CHI '91*, pp. 1-6.
4. Bederson, B. PhotoMesa: A Zoomable Image Browser using Quantum Treemaps and Bubblemaps. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pp. 71-80.
5. Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., & Furnas, G. W. (1996). Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing*, 7, pp. 3-31.
6. Bederson, B. B., Meyer, J., & Good, L. (2000). Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. In *Proceedings of User Interface and Software Technology (UIST 2000)*, pp. 171-180.
7. Bier, E., Stone, M., Fishkin, K., Buxton, W., & Baudel, T. A Taxonomy of See-Through Tools. In *Proceedings of CHI '94*, pp. 358-364.
8. Bier, E., Stone, M., Pier, K., Buxton, W., & DeRose, T. Toolglass and Magic Lenses: The See-Through Interface. In *Proceedings of the 20th Annual Conference on Computer Graphics*, August 1993, pp. 73-80.
9. Chang, B., Mackinlay, J., Zellweger, P., & Igarashi, T. A Negotiation Architecture for Fluid Documents. In *Proceedings of the 11th Annual ACM symposium on User Interface Software and Technology*, 1998, pp. 123-132.
10. Corel Corporation. <http://www.corel.com>
11. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns*. Addison-Wesley Longman, Inc. 1995.
12. The GNU Image Manipulation Program (GIMP). <http://www.gimp.org>
13. IBM VisualAge for Java. <http://www.ibm.com/software/ad/vajava/>
14. Igarashi, T., and Hughes, J. F. A Suggestive Interface for 3D Drawing. In *Proceedings of UIST '01*, pp. 173-181.
15. Kurlander, D., and Feiner, S. A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands. In *Visual Languages and Visual Programming*. S.K. Chang (ed.). Plenum Press, New York, NY., 1990, pp. 257-275.
16. Lunzer, A. Choice and Comparison Where the User Wants Them: Subjunctive Interfaces for Computer-Supported Exploration. In *Proceedings of IFIP TC. 13 International Conference on Human-Computer Interaction (INTERACT '99)*. Edinburg, Scotland, Aug 1999, pp. 474-482.
17. Lunzer, A. Towards the Subjunctive Interface: General Support for Parameter Exploration by Overlaying Alternative Application States. In *Late Breaking Hot Topics Proceedings of IEEE Visualization '98*. Research Triangle Park, North Carolina, Oct 1998, pp. 45-48.
18. Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., & Shieber, S. Design Galleries. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, August 1997, pp. 389-400.
19. Newman, M., & Landay, J. Sitemaps, Storyboards, and Specifications: A Sketch of Website Design Practice. In *Proceedings of Designing Interactive Systems (DIS 2000)*. NY, NY, pp. 263-274
20. Schön, D. A. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, NY. 1983.
21. Stone, M., Fishkin, K., & Bier, E. The Movable Filter as a User Interface Tool. In *Proceedings of CHI '94*, pp. 306-312.
22. Zellweger, P., Chang, B., & Mackinlay, J. Fluid Links for Informed and Incremental Link Transitions. In *Proceedings of the Ninth Annual Conference on Hypertext and Hypermedia*. Pittsburgh, PA, USA. 1998, pp. 50-57.