Selective Undo Support for Painting Applications

Brad A. Myers[1], Ashley Lai[1], Tam Minh Le[1], YoungSeok Yoon[1], Andrew Faulring[1], Joel Brandt[2]
[1]School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{bam, youngseok, faulring}@cs.cmu.edu, {ashleyhlai, minny.le}@gmail.com

[2]Creative Technologies Lab
Adobe Research
San Francisco, CA
joel.brandt@adobe.com

ABSTRACT
Today's widely deployed painting applications use a linear undo model that allows users to backtrack previous operations in reverse chronological order. This undo model is not useful if the user has performed desired operations *after* undesired ones. Selective undo, in contrast, allows users to select specific operations in the past and only undo those, while keeping the remaining operations intact. Although selective undo has been widely explored in the context of text editing and object-oriented drawing, we explore selective undo for painting (bitmap) editing, which has received less attention and introduces many interesting user interface design challenges. Our system, called Aquamarine, explores the script model for selective undo, where selectively undone operations are skipped in the history, rather than the more explored inverse model, which puts an inverse of the selected operations at the end of the history. We discuss the design implications and show through two informal user studies that selective undo is usable and desirable.

Author Keywords
Selective Undo; Bitmap Editor; Creativity Support.

ACM Classification Keywords
H.5.2 User Interfaces (Graphical user interfaces (GUI)); I.3.4 Graphics Utilities (Paint systems); I.3.6 Methodology and Techniques (Interaction techniques).

INTRODUCTION
The *undo* operation has long been understood to be a required command in applications, especially to support creative exploration. Studies have shown that when users have the ability to undo, they are more comfortable exploring and trying new commands [16]. Ideally, any previous operation should be able to be undone. However, most applications use the same *restricted linear undo model* [3], primarily due to the user interface challenges of presenting a more powerful undo model in a way that it is easy for the user to understand. In the linear undo model, multiple operations are saved on the undo stack and can be undone, and after undoing some operations, users can change their mind and start redoing those operations again. However, if a *new* operation is performed after undoing some operations, the undone operations are discarded and are no longer available for redoing. This makes it impossible for the user to undo the undo. Another limitation of the linear undo model is that all operations must be undone in order. This makes it impossible to keep any operations that happen *after* an operation to be undone. That is, if there are a mixture of wanted and unwanted operations, users cannot use undo to achieve their desired result [24].

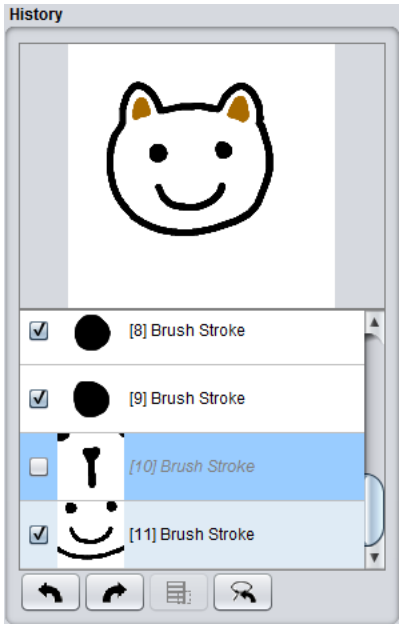

Figure 1. Aquamarine's History Panel showing operation #10 (brush stroke for the nose) selectively undone.

The limitations of linear undo have motivated research on *selective undo*, where the user can specifically select which of the previous operations to undo and which to keep [1-3, 5, 19, 25]. However, this prior work has primarily focused on object-oriented drawing programs, like PowerPoint or Adobe Illustrator [3, 19], where selective undo is more easily achieved by selecting and changing objects' properties. New work has started to address selective undo in text [17, 25], but there is little work on selective undo in painting applications like Photoshop which lack the structures used by object editing programs, and where operations tend to affect overlapping spans of pixels. This makes it much

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea
Copyright 2015 ACM 978-1-4503-3145-6/15/04…$15.00
http://dx.doi.org/10.1145/2702123.2702543.

more difficult for users to identify which operations to undo (since there may be many small edits) and to predict the outcome of selectively undoing something in the past. Furthermore, it is challenging to provide understandable user interfaces for any selective undo model. Note that throughout this paper, we are deliberately distinguishing the term *drawing* from *painting*. Drawing programs have identifiable objects in the picture that can be selected and manipulated later, whereas painting programs convert objects into pixels immediately, and subsequent selections and operations are at the pixel level. Many painting operations cannot be naturally converted into objects with a distinct shape. For example, the paint bucket tool and filters do not create or even necessarily modify objects; instead they manipulate pixels. Aquamarine focuses on painting only. Note that modern image manipulation programs like Photoshop actually have a mixture of drawing and painting features.

Another interesting issue is the distinction between the *inverse* and *script* models of selective undo. In the *inverse model*, supported by most previous selective undo systems [3, 19, 25], a selective undo causes the inverse of the selected operations to be added to the end of the history. For example when the user selectively undoes a delete, a create operation is performed and added to the end of the history. In contrast, the *script model* removes the selected operation from the history and performs all subsequent operations as if that operation had never been performed [15]. This can cause semantic problems in drawing programs (for example, what is the meaning of a change-size operation applied to an object if the previous creation of that object is selectively undone, and thereby removed from the history?), but may be more appropriate when painting (since there are no objects to which operations are applied). We are not aware of any previous research on supporting selective undo in a paint program using the script model.

We created *Aquamarine* (see Figure 1) to study these issues. Aquamarine, which stands for Allowing Quick Undoing of Any Marks And Repairs to Improve Novel Editing, is a new prototype painting program that supports selective undo using the scripting model. We first performed semi-structured interviews with student and professional graphic artists to see what they would want and expect in such a tool. We then implemented the Aquamarine prototype based on those findings, and ran usability evaluations to assess its features and design decisions.

The contributions of this paper include:

- Results of semi-structured interviews, which show that users have a significant need for selective undo capabilities in painting programs, and compensate with workarounds such as using many layers, saving the whole file to disk, or just starting over.
- A discussion of the design space for selective undo and tradeoffs among the design decisions.
- A design and implementation for a selective undo mechanism using the script model in a prototype paint program called Aquamarine, which explores the design space for how the script model should operate.
- A usability evaluation that demonstrates the aspects of the selective undo mechanism which are most usable and useful, and areas where more research is required. All users expressed a desire to have a form of selective undo in their everyday programs.

RELATED WORK

There has been significant work on undo and edit history over the years, in a variety of application domains.

Selective Undo

One of the earliest efforts in this area investigated the script model [2], which tries to guarantee that the final result be as if the prior command had never been performed in the context of text editing for a code editor, and identifies issues specific to that domain. The Emacs text editor has long had an "undo-in-region" command, where the user undoes the most recent operation that affected a specific selected region of text. Other text editors such as DistEdit [21] have provided this feature. Regional undo is useful and also relatively easy to implement compared to the generic selective undo, because it always undoes the most recent operation performed in the selected region, which reduces the possibilities for conflicts. In regional undo, however, there can be an ambiguity if the user selects a region that partially overlaps with an operation's effective region. Li and Li refer to this problem as "region overlapping" and introduce the idea of *partial undo* as a solution, which undoes only the overlapped part of the operation when an operation partly falls in the given undo region [17]. Regional undo was applied in spreadsheets [12] by allowing users to select a region in the spreadsheet and perform undo only on those cells. Our new system, Azurite [24, 25], provides general selective undo (not just region undo) in a code editor. Azurite uses the inverse model, by adding the inverse text editing operation to the end of the history, and provides a variety of user interfaces to resolve conflicts.

Probably the most work on selective undo has been in the context of graphical (object-oriented), interactive editors. Berlage introduced the selective undo model that adds the reverse operation of the selected command to the current context [3]. The Amulet [20] and Topaz [19] systems had a similar selective undo feature, but these allowed repeating a selected command even on a new object. All of these use the inverse model, with the undos added to the end of the history. Dwell-and-spring [1] provides an interface that is limited to undoing only press-drag-release interactions, and it does not deal with any conflicting operations. All of these approaches also assume that there is an object on which the operations can be performed: primitive graphical objects such as shapes in graphical editors, and individual cells in spreadsheets. In contrast, there is no clear notion of objects

in painting programs, since edit operations typically affect areas of pixels.

Another important difference between object-oriented and painting systems is that detecting and handling conflicts and dependencies among the operations is important for all object-oriented selective undo systems. Whereas early systems using simple dependency models [3, 19], later work focused on formal, sound and complete modeling of dependencies, and even automatically cascading the selective undo when necessary for consistency [6]. User studies showed that people could predict and understand what these selective undo systems will do [7]. Whereas in one sense, painting systems do not have such dependencies since all actions operate on pixels, we do study the issue of conflicts when the boundaries of actions overlap, as discussed below.

There are other undo models that support selective undo by providing additional commands beyond undo and redo. The US&R model [22] allows users to *skip* redoing an operation, using a complicated tree-based data structure. Users can selectively undo an isolated operation by undoing multiple steps until the target operation gets undone, skipping the redo command once, and then redoing the rest of the operations. The triadic model [23] uses a simpler structure composed of a linear history list, and a circular redo list which can be *rotated* by users. Undoing an operation puts the operation at the beginning of the redo list, and rotating the redo list takes one operation at the beginning of the list and puts it at the end. Since the rotate command can be used to skip a redo command, users can selectively undo a certain operation in a similar way. However, both models require deep understanding of the underlying history structure to correctly perform selective undo. In addition, selective undo cannot be done in one step, which can be cumbersome for users.

Adobe Photoshop provides a history window with a mode for "non-linear undo", but this is different from selective undo—when the user undoes operations and then does new ones, Photoshop's non-linear undo retains the undone operations on the undo stack rather than remove them. However, future undos still start at the last operation and continue backwards through all previous operations in order.

Another use of selective undo is in collaborative editing, where multiple people can edit a document concurrently, which has been studied by various systems [4, 10]. When user1's and user2's edits are intermixed, it is not clear whether user1's undo command should affect user2's operations. Surprisingly, the Google Docs text editor does not seem to do anything sensible when multiple people edit in the same region and then one performs undo.

A revision control system for images based on a directed acyclic graph (DAG) enables users to make forks and joins, and then move around in the history and see various versions of images [8], but it does not support selective undo or the script model.

Graphical Edit Histories

Aquamarine displays the past interactions in a graphical History pane (see Figure 1). There has been significant research on such displays. Chimera provided graphical histories as thumbnail snapshots which could be edited and reused, and past actions could be modified [15], but conflicts among operations were not specifically identified. The Designer's Outpost shows snapshots of the history of states of a web editing session with multiple users and keeps track of forks among versions [13]. Systems have also used graphical histories to foster learning [9, 11] and creating macros for later reuse [15, 18].

INITIAL SEMI-STRUCTURED INTERVIEWS

In order to get a better idea about how people might use a selective undo mechanism in painting applications, we first performed semi-structured interviews of nine people. Our last author from Adobe reports that there are three primary uses of Photoshop: (1) to design and prototype user interfaces, (2) as a painting tool to create art, and (3) to edit and retouch photographs. We were careful to recruit participants from all three groups (see Table 1).

ID	Experience	Student or Prof.	Des?	Art?	Photo?	Ctrl-Z	Step-Back	Hist. Panel	Hist. Brush
1	Intermediate	student	Yes	Yes			Yes	Yes	Yes
2	Intermediate	student	Yes	Yes		Yes			
3	Intermediate	student			Yes			Yes	
4	Expert	student	Yes			Yes	Yes		
5	Expert	prof.		Yes		Yes	Yes	Yes	
6	Novice	student		Yes			Yes		
7	Expert	prof.	Yes				Yes		
8	Expert	prof.	Yes				Yes		
9	Expert	prof.	Yes				Yes		

Table 1. Participants in our semi-structured interviews.

All of our participants were frequent users of Photoshop, with four self-rating as expert users, four as intermediate and one as novice. Four were professionals and five were students. Photoshop provides four undo mechanisms: an old-fashioned toggle undo (Ctrl-Z), which undoes and then redoes the last operation, a normal linear undo, which it calls "Step Backwards" and "Step Forwards" (Shift/Alt-Ctrl-Z), a history panel which enables undoing back multiple steps (with a fixed maximum limit that defaults to 20), and a "history brush" where users select a point in the history panel and then paints, which restores the pixels back to that point in time, as a form of regional undo. As previously mentioned, Photoshop provides an optional non-linear history mode. No participant (in this or the final study) had ever used or had even heard of this feature, which is turned off by default. Three of the participants reported using the toggle undo, five reported using step-backwards and step-forwards, three used the history panel, and one reported using the history brush.

In the first part of our interviews, we asked participants to work on one of their own tasks using Photoshop, in a form of contextual inquiry, and they were encouraged to think aloud. We told them we were particularly interested in situ-

ations where they would "explore different ideas and test out different alternatives."

The more experienced users exclusively used the shortcut keystrokes to invoke commands. Only two participants had the history panel displayed, but both of them used it to undo multiple operations. We did not see any use of the history brush. Some participants erased areas rather than using undo, whereas others made significant use of the step-backwards commands. They recognized the limitations: "I don't want to use the undo button if there's a part that I drew after that I like." Another said: "It's more of a short-term memory [issue] for me. I don't usually undo more than five steps back."

Participants used the layers mechanism to group items together when they anticipated a need to selectively change an area or a shape, and experienced users grouped and named layers to keep the large number of layers organized. When beginning a creative exploration, participants often duplicated layers or saved a version of the whole picture, to facilitate backtracking. Users often hid layers instead of deleting them; one said: "I don't like deleting things; it feels too permanent—even if it was a mistake."

In the second part of our interviews, we gave the participants a series of tasks designed to elicit selective backtracking behaviors and strategies. We told them to imagine they were creating images for a company that is run by a very indecisive boss. We would repeatedly instruct them to create something, then change it, then do other things, and finally to change it back. This cannot be supported with any of Photoshop's existing undo mechanisms. As in the first part, when participants anticipated that something would need to be undone, they put it on a separate layer. If we asked them to undo something that was not on a separate layer, they expressed regret that it was not separated and usually started over from scratch. Users considered using advanced editing tools such as Photoshop's Color Replacement Tool, but often decided it would not work sufficiently well, and just painted over items, or deleted and redrew them from scratch. These results reveal the reliance on layers to simulate selective undo, and the need to start over and lose desired work when undesired operations precede the desired ones.

In the third part of our interviews, we explained the concept of selective undo and asked if they thought it would be useful in their own work. Six participants said it would be useful frequently or almost every day, and the other three said maybe not since they achieved the same functionality with layers or by frequently saving their work to disk.

In the fourth part of our interviews, we asked a series of questions about what they thought would happen as a result of performing a selective undo. In particular, we were exploring whether they expected the *script model* or the *inverse model* of behavior. For example in Figure 2, if the stars were created by duplicating the original star, what should happen if the creation of the original star was selectively undone? All participants thought the other stars should *not* disappear, so they were endorsing the *inverse* model in this case, since under the script model, there would be nothing to duplicate. On the other hand, if we added a recolor step for the star between steps 1 and 2, and then selectively undid the recoloring, all but one of the participants expected the stars in step 2 to also change color, requiring the script model to hold. When this inconsistency was pointed out, most participants agreed that their preference would change based on the situation.

Finally, we explored a number of features that might be used in a selective undo mechanism. Most participants agreed that the ability to identify which operations had contributed to a selected region of the picture would be useful, as a way to find which operations to selectively undo. About half thought the reverse operation was needed – to select an operation in the history panel and highlight what area of the picture it affected. Few participants saw any need to search for commands in the history by name (to enable searches like "find the last time I used the brush tool"). Also deemed unnecessary would be a need to view different versions of a picture side-by-side, or automatic version control tools, as are commonly used for source code and which are provided by third party tools like LayerVault (layervault.com) and Pixelapse (www.pixelapse.com).

Discussion

In an object-oriented drawing program, users can generally get the effect of undoing operations by manually performing the opposite operation. Thus, the "undo" of resizing can be achieved by just resizing it back, which can generally be performed at any time. Photoshop and other painting programs try to provide this capability for as many operations as possible by having many operations act like they are object-oriented instead of pixel based (that is, like a drawing program instead of a painting program). For example, text in Photoshop is kept as objects on its own layer until the user explicitly flattens it into a bitmap. However, this means that many bitmap operations, like blurring or erasing, would not be available until the image is flattened, and changing the text would not be possible after blurring.

Similarly, in our study, we saw participants using layers to try to make painting operations have the ability to be selectively edited. However, layers do have significant limitations. We saw frequent errors where participants would accidentally draw into the wrong layer, merge layers and regret it later, or realize too late that a new layer should have been created. Also, the large number of layers became difficult to manage.

Figure 2: Multiple steps to create a drawing.

In addition, users were wary of the Photoshop history tool, since it only keeps a limited number of steps backwards, the history is not preserved when a document is closed, and it can be difficult to identify which step to go back to. One participant noted: "If you rely on history, you're asking to get burned by it." Therefore, there is an opportunity to provide a new way for users to backtrack.

Participants agreed that there would be significant value in having a selective undo tool that would provide the ability to change what was done in the past, without the requirements of pre-planning to use layers and without giving up the advantages of bitmap editing.

DESIGN TRADEOFFS

In designing and implementing the selective undo feature, we had to make a large number of design decisions, across a number of different dimensions. In many cases, we needed to decide what a selective undo for painting applications should mean since this has never previously been explored. This section presents the design dimensions, the various options and tradeoffs, and justifications for decisions implemented in Aquamarine.

Which Operations to Explore?

We classified all Photoshop operations into eight categories with respect to their interaction with Undo (as has also been explored previously by others [8]). The categories are:

1. Creation/painting, which cause pixels to be drawn
2. Local-adjustment tools, which change existing pixels
3. Global commands, which affect all pixels in the image
4. View control commands, which do not change pixels
5. Selection tools, which control subsequent operations
6. Mode changes, like picking a color or current layer
7. Conversion tools, like Flatten and Convert to Smart Object
8. Miscellaneous, like 3D tools

The first three are most relevant to selective undo, so we implemented at least two operations from each of these categories. We expect that operations in Photoshop or other full-featured paint programs would have identical issues with respect to selective undo, so we believe the HCI issues we have identified will generalize to all other commands.

Script Model versus Inverse Model?

The most important decision was which kind of selective undo model to support. As discussed above, the primary choices are to use some form of script model or some form of inverse model.

The inverse model seemed inappropriate for a painting program, since so many operations cannot be undone by adding a new command to the end of the history. For example, take color change. In a painting program, a color change using the Paint Bucket Tool (more formally called flood fill), changes the color of the area defined by the contiguous pixels that match, within a tolerance, the color of the clicked pixel. This operation often *cannot* be undone with a new flood-fill at the end of the history (that is, "now"). For example, in Figure 3, the flood fill of step 2 cannot be undone with another flood fill after all of the actions in step 3, because flood-filling now will only change parts of the former blue t-shirt, and the long sleeves added to the shirt might also be flood-filled. Similarly, other bitmap editing operations, like blurring the image, cropping, etc. may be impossible to reverse in the current state. Other operations, even creating new shapes, cannot necessarily be undone in the current state, because of other paintings drawn on top of them may compute the new pixels using the existing pixels.

However, a script model *can* be used to selectively undo the flood fill operation—we can undo all the operations back to before step 2 was done (so the image looks like step 1), skip doing step 2, and then do the remainder of the operations.

Note that using layers to separate operations as in Photoshop and other programs, so the user can simulate selective undo by turning off or deleting layers, will not help with selective undoing of painting-based operations like flood fill or blur, since these operations must work on the same layer as the pixels to be modified.

Since Aquamarine specifically focuses on these painting operations, we decided to use the script model. Another motivation is that the script model has rarely been investigated in previous research or commercial systems, and it brings up many more interesting and challenging design decisions, which we discuss next.

Note that providing selective undo with either model can work with the regular linear undo command. Thus, if the user does not use selective undo, it can be ignored entirely, and the normal way of using undo/redo can be utilized.

Another issue is that some commands have side effects *external* to the editor. For example, File Save should not be re-executed each time the system reruns the script to perform a selective undo. Fortunately, it turns out that these operations are actually not put into the undo stack anyway (see also the discussion of copy and paste below). For a few *internal* operations with side effects, like Create Layer, we must disable all the other operations on that layer if the create is selectively undone.

Handling Region Conflicts among Operations

As discussed in the related work section, an important concern in object-oriented editing is dealing with dependencies and conflicts [3, 6, 19]. Bitmap operations do not have this kind of conflict, since they work on pixels, which will still

Figure 3. In a painting program, (1) paint a shirt, (2) flood fill it with a new color, (3) then do a variety of other actions.

be there. However, it is possible to have *region conflicts* [25], which are where a later operation's scope overlaps the selected operation in such a way that it is not clear what the selective undo should do. For example, in Figure 3, if the user tried to selectively undo the painting operation (step 1), what should the flood fill (step 2) do, since the t-shirt is no longer on the screen?

There are a variety of possible outcomes for step 2 in a script model when step 1 is removed:

1. Cancel the selective undo due to the conflict. That is, do nothing.
2. Perform the selective undo, and also undo the conflicting operations, which includes at least step 2.
3. Perform the conflicting operations as dictated by the script model with the selected operation removed. In this case, it would probably flood-fill the whole canvas since there would be nothing (only background) at the pixel where the flood fill operation happened.

There is actually a 4th possibility, which we discuss in Future Work: allow the user to *modify* the conflicting operation or *add* entirely new operations into the history. For example in Figure 3, the user might select a new pixel at which to apply the flood fill, or paint a brand new shape where the shirt used to be as a new step 1.

We investigated providing a popup dialog to help the user resolve these conflicts and pick one of the three options, as in Azurite [25], but it proved quite difficult to identify exactly which operations should be marked as conflicting with the undone operation, resulting in very expensive pixel operations and too many false positives. Therefore, we decided to use a much simpler approach of just comparing bounding boxes of all subsequent operations, and highlighting the operations in the history which *might* conflict, to help the user find them if they want option 2 (see Figure 4).

To reduce the number of operations that are highlighted, we developed a set of heuristics for the kinds of region conflicts that should be brought to the user's attention based on the operation categories discussed above. The heuristics are as follows: first, we do not highlight creation/painting operations. For example, if a new shape is painted on top of an old shape, and the painting of the old shape is selectively undone, even though overlapping pixels are affected, we do not alert the user, and simply remove the old shape and repaint the new shape. Similarly, global operations affect all pixels, but we do not alert users to these region conflicts either. The main issue is when the later operation is a *local-adjustment tool*, which modifies only some of the pixels of the picture. Flood-fill and smudge are examples of this, and there are many others. When such operations affect the same area that the selective undo will affect, we alert the user of all conflicting operations using orange highlights.

As future work, we plan to support allowing the user to select *multiple* operations to be selectively undone together [25]. For example, the user might select both Steps 1 and 2 together in Figure 3. When all the region conflicts are internal to the selected operations, there is no ambiguity of what to do, so the user does not have to be alerted. In our usability evaluation, this was a much-requested feature.

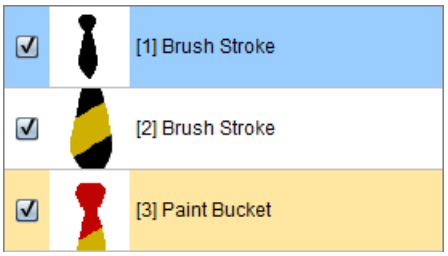

Figure 4. Highlighting operation 3 in orange since it conflicts with the selected operation 1 (shown in blue).

Copy and Paste

Another interesting design issue revolves around copy and paste. In most existing applications, the copy operation is *not* put on the undo stack, and whatever is copied is retained independent of what the user subsequently undoes. For example, in Microsoft Word, a user can type some text and copy it, and then invoke undo one or more times so that the text is all removed. However, the clipboard will still retain the text, which can be pasted later. Of course a cut operation goes on the undo stack, but only the deleting part of the cut is undone—again the clipboard is not affected by undoing the cut operation. Users have developed strategies for clever ways to use this feature, so we decided to retain it in Aquamarine.

Therefore, Aquamarine makes what we call a *deep copy* of whatever is selected when the user performs a copy operation, and the copy operation is not put into the undo stack. That means that the script model will not re-execute the copy operation, so the clipboard will continue to have a copy of the pixels as they were originally copied, no matter what happens subsequently to that part of the image. Similarly, the paste operation retains a deep copy of what was in the clipboard when the operation is first invoked, so it continues to paste the *same* picture if it is reinvoked later due to the script model.

Another reason we felt that we needed to adopt this model for copy is that the clipboard is globally shared by all programs, and the user might change the clipboard by doing a copy in *another* program, and in that case, we would not want to change the clipboard as a result of a selective undo.

Selective Undo/Redo Operations in the History Panel

In Topaz [19] and Azurite [24] and many other systems that implement the inverse model, the selective undo and redo operations themselves are added to the end of the undo history, and shown in the history panel (see Figure 5). This seems to make sense in those programs since the inverse operation is actually performed at that point in the history, and so a representation like Figure 5 should reinforce this mental model for users. Another advantage of this approach is that the selective undo operation itself can be selected and undone, which would then add another selective undo

to the end of the history, restoring the effect of the original operation. Alternatively, the user could select the original operation (#5 in Figure 5) and selectively *redo* that operation, which has the same effect as selectively undoing the undo. A final advantage of this design is that it is clearer what a regular linear undo command will do – it works up from the bottom, undoing each of the operations in turn (selective and regular operations alike).

Given these advantages, we first tried adapting this style of history panel for the script model of Aquamarine, as shown in Figure 5. In this design, the user can select an operation (e.g., #5), and invoke selective undo on it, which adds the selective undo operation to the end of the history (shown as operation #7). Unlike in the inverse model, the selective undo is not really on the history, but instead the referenced operation is skipped (in Figure 5, only operations 1 through 4 and 6 are executed). We try to visualize this by graying out the referenced operation. If the user wants to reverse this operation, the selective undo (#7) can be selectively undone or the original operation (#5) can be selectively redone, and these operations would be also added to the history. However, we felt it was getting complicated to understand what was done and not done, and we also felt that this design might give the wrong mental model to users, since the selective operations are not really in the history.

Therefore, we designed a different history panel, shown in Figure 1, where the original operations can simply be unchecked to selectively undo them, or rechecked to be selectively redone. This matches the design for the layers panel in Photoshop, where layers can be made visible or not with a toggle button next to each layer's name. This should do a better job of matching the appropriate mental model for users: that the current picture results from the operations being executed from the top down, skipping operations that are not checked. Another advantage is that we do not need explicit selective undo or selective redo commands, but instead the user just toggles the appropriate checkbox.

It is less clear what the regular linear undo should do in this model. We decided linear undo should undo the most recent operation shown in the history panel that is still in effect. Therefore, the selective undo is not directly undoable by Ctrl-Z. This again mimics the default behavior of turning on and off layers, which are not put on the history stack (although Photoshop has a setting to enable this). Similarly, the regular linear *redo* moves down the history stack re-enabling operations, no matter how they were undone. Thus, it redoes operations that were disabled either by selective undo or regular undo. Of course, the user can always go to any individual operation and toggle its checkbox, no matter how it was disabled.

Since the "right" design for this feature is not clear, we decided to include questions about this in our usability evaluation, discussed below, which came out strongly in favor of the checkbox version shown in Figure 1.

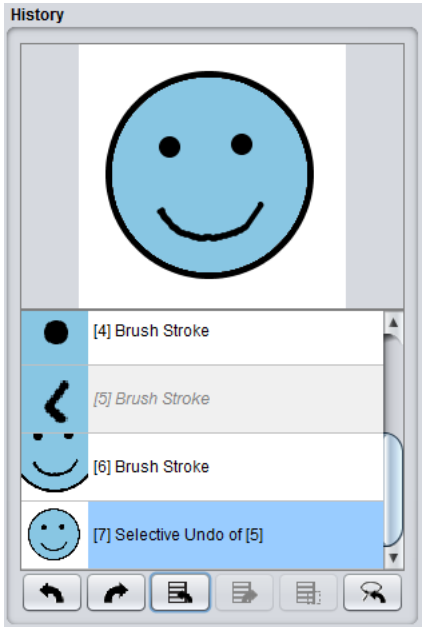

Figure 5. An alternative form of history panel where selective undo/redo operations are included in the history.

Thumbnail Images

One small design issue is: what to display for each operation in the history panel. A common complaint about Photoshop's history panel from our initial semi-structured interviews was that it only shows an icon for the tool used, so it is impossible to tell which operation is for which part of the drawing, for example when there are many brush strokes. Therefore, we wanted to show a thumbnail representation of what the operation did, along with some context, as has been done previously [15].

A new complication for Aquamarine that has not been previously reported arises from the scripting model of selective undo—what should be shown in the thumbnail when a selective undo causes a previous operation to be turned off? Should the thumbnails of the subsequent operations be changed? We decided to update all the thumbnails, since that is the more correct (and interesting) design. Ironically, in the usability evaluation, users almost universally said they did *not* want *any* context to be shown in the thumbnails, completely eliminating this problem. Instead, they preferred seeing only the specific output of the current operation. The preview window (the upper panel of Figure 5), however, should continue to show what the complete picture would look like up to the selected operation.

Identifying Desired Operations

Azurite [24] provides elaborate ways to *search* for the operations that the user might want to undo. However, in our initial semi-structured interviews, the key way that users wanted to search for operations in a painting program was by selecting an affected region on the screen. Therefore, we provide commands in Aquamarine for identifying the set of operations that affect the selected region on the screen. We

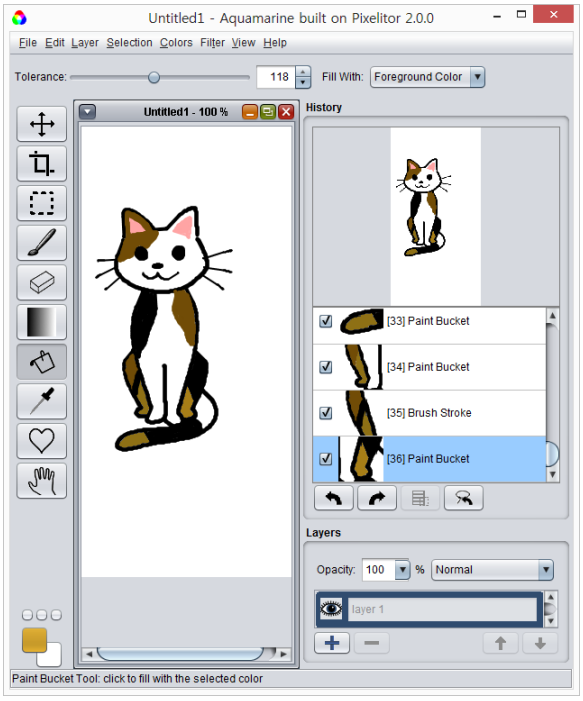

Figure 6. Pixelitor modified with our history panel.

also provide the reverse operation which shows the region on the screen that any operation affects. Note that both of these are based on the specific pixels on the screen, which may no longer hold that operation's result. For example, if the user paints a shape, but later selects and moves it, the region for the original painting operation will remain where the shape started. In a painting program, it seems impossible to identify where the pixels that result from an arbitrary operation might have gone, and it seems more useful to help users find where things used to be, in case they want to get back to that state. Note that if the user selects the region on the painting where the shape was originally painted, both the painting and the move operations will be identified, so users can selectively undo/redo any operations that are desired.

IMPLEMENTED SYSTEM

Our original hope was to implement Aquamarine as a plugin to Photoshop since it supplies a variety of APIs. Unfortunately, none of Photoshop's APIs provided sufficient access to the undo stack and the result of each operation to support the desired features. Therefore, we needed to find an open-source paint program we could modify. We required sufficient functionality so that the design issues discussed above would emerge (which eliminated simple painting programs like the Java Paint program used in other studies [14]), and we needed access to the full source to be able to implement the selective undo.

We selected Pixelitor 2.0.0 in Java by László Balázs-Csíki (http://sourceforge.net/projects/pixelitor/) (see Figure 6). The original version of Pixelitor used the built-in Java Swing undo model, which uses command objects [20]. We were able to replace this with our own undo implementation which supports our novel selective undo mechanism.

One complication is that Pixelitor, like other painting programs including Photoshop, implements undo by simply having most operations save a bitmap of the picture before the operation executed. This is sufficient for the regular linear undo model, since it can always be sure that when it places the bitmap into the correct position, that the context will be correct and the full image will be restored. However, this is insufficient for any selective undo model, especially for the script model, since we need to be able to re-execute all commands later. Therefore, we had to modify each command to remember all of its parameters so it can be re-executed when needed. For example, the Shape and Brush stroke tools must save the path and all the properties of the pen (color, transparency, etc.). Similarly, the Paint Bucket tool needs to remember the color, start pixel, etc. Further, we had to refactor the application so all operations could get their parameters from the global widgets (like the current color) or from the saved command objects. We still save the bitmaps so they can be used for regular undo.

Another implementation tradeoff was whether each operation should save the entire bitmap of the whole picture, or only the portion affected by this operation. If each operation saved the entire picture, then our script model could implement the selective undo of an operation by simply reinstating the full picture before that operation, skipping the operation, and then re-performing all enabled subsequent operations. However, saving the entire bitmap wastes a lot of memory, since most operations only affect a tiny part of the picture. Pixelitor only stores bitmaps for the regions affected by each operation (which it uses for undo), which we retain. Therefore, implementing the script model selective undo does not require any significant amount of extra space. The tradeoff is that for selective undo, we must internally roll back to the selected operation, by doing a linear undo all the way back, in order to restore the picture to the original state. This is a classic space-time tradeoff, and in practice, we found it to be sufficiently fast. If it turns out not to be fast enough in the future, there are many obvious optimizations that could be implemented.

Some Photoshop operations can be quite slow and expensive, so there is a question about the efficiency of re-applying all the operations each time a selective undo is invoked, which is required for the script model. In our unoptimized prototype implementation, selective undo slows down noticeably on a big image if there are over 100 operations. Although we focus on the user interface aspects here, we have considered some performance optimizations that could be applied in a real implementation. Obviously, the length of the history can be limited, as is already the case in Photoshop, but that is not desirable from a UI standpoint. Instead, slow operations could be approximated on downsized images, to show quick previews which estimate what

the results will be, with the real result calculated in the background once the user stops undoing and redoing.

USABILITY EVALUATION

Since our initial semi-structured interviews and other prior work had clearly identified problems that our mechanism addresses, we felt the key issue to evaluate was whether users could understand and successfully apply our new script-model selective undo, and whether users can think of strategies that would make effective use of it. Therefore, we felt the appropriate evaluation would be to do think-aloud usability evaluations of various versions of the features which embody the design decisions discussed above.

We recruited eight participants (see Table 2), one of whom was also part of our initial semi-structured interviews (participant 5 in Table 1 is participant 2 in Table 2). We specifically picked users across a wide range of experience. None of the new participants were students. Four were professional graphic artists and the other four were professionals who only used Photoshop or equivalent occasionally. All participants were familiar with conventional undo models, as supported by Photoshop and most applications, but three participants were not familiar with Photoshop's special undo features.

ID	Experience	Prof.	Des?	Art?	Photo?	Ctrl-Z	Step-Back	Hist. Panel	Hist. Brush
1	novice	other			Yes	Yes	Yes		
2	expert	prof.		Yes		Yes	Yes	Yes	
3	expert	prof.	Yes			Yes	Yes	Yes	
4	expert	prof.	Yes	Yes		Yes	Yes	Yes	
5	intermediate	prof.	Yes			Yes	Yes	Yes	
6	novice	other	Yes			Yes	Yes		
7	intermediate	other	Yes	Yes		Yes	Yes		
8	expert	other			Yes	Yes	Yes	Yes	

Table 2. Participants in our usability evaluation.

We asked the participants to "try drawing a few things" with Aquamarine, and then undo operations using a variety of commands. Most participants just drew random lines and shapes, but p5 drew animals like Figures 1 and 6. We then asked them what they would expect to happen with various undo commands (regular Ctrl-Z and our new selective undo) and then to try them out to see what happens. We had about half of the participants use the checkbox version of the history first (Figure 1), and the other half tried the in-history version first (Figure 5). Then they tried the other version. The order had no effect, and *all* 8 participants strongly preferred the checkbox version. Everyone understood the operation of the checkboxes and how they would affect the picture, although some expressed a preference to using an "eyeball" icon as used to toggle layer visibility (see Figure 6). In the in-history version, they found the presence of the "selective undo" and "selective redo" operations in the history to be confusing, and also felt that this would clutter up the history without being useful.

All users found the script model to be understandable and preferable in general, but all users were surprised by the operation of the paint bucket, when the area underneath where it was applied was selectively undone (a situation in which there is a region conflict). In this case, the paint would typically fill the whole background, which was not something anyone wanted. Some participants wanted the paint bucket to just remember the shape it had filled and reuse that, even if the area was no longer there, but most people wanted the system to be smarter about undoing the paint bucket operation with the previous operation. In discussions, however, they agreed this would be tricky in practice, and that the orange highlighting of the conflicting operations (Figure 4) would be a reasonable way for users to manually find and fix conflicts.

Interestingly, all but one participant was surprised by the operation of copy-and-paste, even though everyone was familiar with the way copy-and-paste works in other programs (where the copy ignores undo). However, participants agreed that our design was consistent and would sometimes be useful.

The main requested feature was a way to *group* actions. For example, selecting multiple operations together and undoing them all at once, or collecting operations into named groups, like layers can be in Photoshop.

In summary, the evaluation showed that with the check-box version, the script-model selective undo was understandable and usable, and that people understood how the issues from the semi-structured interviews would be addressed by our tool. All participants expressed a strong desire to have this kind of selective undo in Photoshop, and even in their other editing programs. They said that selective undo could substitute for some of the ways they now use layers.

FUTURE WORK

Currently, Aquamarine is an early prototype, sufficient to explore the design issues discussed here, but not yet ready for deployment. The current work includes making the rest of the features of Pixelitor work with selective undo, and then releasing Aquamarine as open source for general use and a full field test.

Photoshop has an option to add the operations that enable and disable layers onto the undo history, and Topaz [19] even could put changes of selections and the "find" operation into the undo history. Some of our participants expressed the desire to have state changes that do not affect the current painting, such as changing the current color or the radius of the brush, included into the history so they could be undone. We propose to explore these in the future.

We also want to investigate how to allow operations in the past to be *modified*. For example, some participants in both studies expressed a desire to be able to change the color of an operation done in the past, and see that propagated through the rest of the edits. The next step is to enable *new* operations to be inserted into the past, or existing operations to be reordered, for example to put something behind other painting in the stacking order. Although other research sys-

tems have tried some of these [15], a key usability challenge remains of how the user would be able to understand, undo and selectively undo changes to the history.

Another requested feature from our participants was some way to collapse long histories. For example, text editors coalesce multiple keystrokes into a single undoable action, and we would like to explore the ability to collapse and expand multiple small brush strokes. This might be supported both automatically and manually, so the user could achieve the desired level of granularity when navigating.

Finally, we would like to explore saving and restoring histories. The history might be stored in the image document, to enable cross session undoing. This could also enable someone later to explore how an effect was achieved. Similarly, a saved script could be converted into a tutorial [9]. Sections of a history could also be selected and converted into parameterizable reusable macros, as in Topaz [19].

CONCLUSIONS

Providing selective undo in a painting program brings up a surprising number of design challenges. Aquamarine shows that UI research can address those challenges and usable interfaces can be provided that can have benefits for users. We hope this research will inspire others to investigate providing selective undo in a variety of domains.

ACKNOWLEDGEMENTS

We thank Min Jeong Kim for her help with the figures used in this paper. Funding for this research comes in part from NSF grant IIS-1314356, from the Korea Foundation for Advanced Studies (KFAS), and from Adobe Research. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF, KFAS, or Adobe.

REFERENCES

1. Appert, C., Chapuis, O., and Pietriga, E., "Dwell-and-Spring: Undo for Direct Manipulation," in *SIGCHI'2012*. Austin, TX. pp. 1957-1966.
2. Archer Jr., J.E., Conway, R., and Schneider, F.B., "User Recovery and Reversal in Interactive Systems." *ACM Trans. Program. Lang. Syst.*, 1984. **6**(1): pp. 1-19.
3. Berlage, T., "A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects." *ACM Trans. on Comp. Human Inter.*, 1994. **1**(3): pp. 269-294.
4. Berlage, T. and Genau, A. "A Framework for Shared Applications with a Replicated Architecture," in *ACM UIST'1993*. Atlanta, GA: pp. 249-257.
5. Cass, A. and Fernandes, C., "Using Task Models for Cascading Selective Undo," in *Inter. Conf. on Task Models and Diagrams for UI Design*, K. Coninx, et.al, Eds. 2007. Springer Berlin / Heidelberg: pp. 186-201.
6. Cass, A.G. and Fern, C.S.T., "Modeling Dependencies for Cascading Selective Undo," in *IFIP INTERACT 2005 Workshop on Integrating Soft. Eng. and Usability Eng*.
7. Cass, A.G., Fernandes, C.S.T., and Polidore, A., "An Empirical Evaluation of Undo Mechanisms," in *NordicCHI*'2006. Oslo, Norway. pp. 19-27.
8. Chen, H.-T., Wei, L.-Y., and Chang, C.-F., "Nonlinear Revision Control for Images." *ACM Trans. Graph.*, 2011. **30**(4): pp. 1-10 (Article 105).
9. Chi, P.-Y.*, et al.*, "Mixt: Automatic Generation of Step-by-Step Mixed Media Tutorials," in *UIST'2012*. ACM: Cambridge, MA. pp. 93-102.
10. Choudhary, R. and Dewan, P., "A General Multi-User Undo/Redo Model," in *ECSCW'1995*. Springer Stockholm, Sweden. pp. 231-246.
11. Grossman, T., Matejka, J., and Fitzmaurice, G., "Chronicle: Capture, Exploration, and Playback of Document Workflow Histories," *UIST'2010*. pp.143-152.
12. Kawasaki, Y. and Igarashi, T., "Regional Undo for Spreadsheets (Demo)," in *Adjunct Proceedings UIST'2004*. 2 pages.
13. Klemmer, S.R.*, et al.* "Where Do Web Sites Come From?: Capturing and Interacting with Design History," in *CHI'2002*. Minneapolis, Minnesota: pp. 1-8.
14. Ko, A.J.*, et al.*, "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software Maintenance Tasks." *IEEE Trans. on Soft. Eng.*, 2006. **33**(12): pp. 971-987.
15. Kurlander, D. and Feiner, S. "Editable Graphical Histories," in *1988 IEEE Workshop on Visual Languages.* Pittsburgh, PA: pp. 127-134.
16. Kuttal, S.K., Sarma, A., and Rothermel, G., "History Repeats Itself More Easily When You Log It: Versioning for Mashup," in *IEEE VL/HCC'2011*. pp. 69-72.
17. Li, R. and Li, D., "A Regional Undo Mechanism for Text Editing," in *Inter. Workshop on Collaborative Editing Systems*, 2003.
18. Lieberman, H. "Dominos and Storyboards: Beyond Icons on Strings," in *1992 IEEE Workshop on Visual Languages*. Seattle, WA: pp. 65-71.
19. Myers, B.A. "Scripting Graphical Applications by Demonstration," in *SIGCHI'1998*. pp. 534-541.
20. Myers, B.A. and Kosbie, D., "Reusable Hierarchical Command Objects," in *CHI'1996*. pp. 260-267.
21. Prakash, A. and Knister, M.J., "A Framework for Undoing Actions in Collaborative Systems." *ACM Trans. on Comp.-Human Inter.*, 1994. **1**(4): pp. 295-330.
22. Vitter, J.S., "Us&R: A New Framework for Redoing (Extended Abstract)." *SIGSOFT Software Engineering Notes*, 1984. **9**(3): pp. 168-176.
23. Yang, Y., "Undo Support Models." *Inter. J. of Man-Machine Studies*, 1988. **28**(5): pp. 457-481.
24. Yoon, Y., Koo, S., and Myers, B.A., "Visualization of Fine-Grained Code Change History," in *IEEE VL/HCC'2013*. pp. 119-126.
25. Yoon, Y. and Myers, B.A., "Supporting Selective Undo in a Code Editor," in *ICSE 2015*, Florence, Italy, to appear.