Editable Graphical Histories

David Kurlander
Steven Feiner

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

Graphical interfaces typically provide their users with little idea of a session's history, except insofar as it is reflected in the current state of the system. If undo and redo commands are provided, they are often the only way to review the actions performed, cycling through them in sequence. We introduce the notion of an editable graphical history that can allow the user to review and modify the actions performed with a graphical interface. We have designed a testbed system that creates a series of automatically-generated panels that depict in chronological order the important events in the history of a user's session with Chimera, a graphical editor. Our system uses heuristics to determine the contents of each panel and the actions that it illustrates. The user may scroll through the sequence of panels, reviewing actions at different levels of detail, and selectively undoing, modifying, and redoing previous actions.

Keywords: visual languages, editable graphical histories, undo, redo, timeline display, user interface design, graphical editing

1. Introduction

As a session with a computer system unfolds, it is natural for the user to want to review previous actions, to undo or redo in part what has been accomplished, and even to reapply modified versions of previous commands. Consequently, a number of user interface techniques have been developed that provide a sense of the session's history. For example, command histories, typified by that of the UNIX™ command shell *csh* [JOY79], allow the user to list, edit, and re-execute previously executed commands. Many window-based systems further provide the ability to scroll over the entire transcript of a session and select text for re-execution [SWIN86]. We will refer to this approach as *spatial browsing*. Spatial browsing relies on an implicit model of a session, or at least of the commands executed during it, as a continuous scroll that can be browsed spatially from beginning to end.

This work is supported in part by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0165, the New York State Center for Advanced Technology under Contract NYSSTF-CAT(87)-5, and by an equipment grant from the Hewlett-Packard Company AI University Grants Program.

In contrast, most graphical interfaces and some textual interfaces, such as full screen text editors, do not share this virtual scroll model. These systems often attach importance to the actual physical screen locations at which input and output occur, modifying in place what is being displayed. Examining the session history in such systems often involves what might be called *temporal browsing*, in which commands are "played back" to show their effect on the system's state. The ability to record all user actions and replay them to recapitulate a session's history has been an important feature in graphical interfaces such as those of computer paint systems [SMIT78]. Similarly, undo and redo commands of the sort often implemented in editors typically result in the screen being partially redrawn to show the state before or after a command was executed. These techniques are most commonly used for making modifications to past actions, rather than merely browsing.

There have been several attempts to merge the spatial and temporal history browsing paradigms for graphical interfaces. Feiner, Nagy, and van Dam's IGD hypermedia system [FEIN82] allows readers of its graphical documents to view an automatically constructed timeline displaying time-stamped miniatures of the pages displayed during the session. Readers can scroll through the timeline display, selecting a previous page to return to if desired. Christodoulakis and Graham [CHRI88] describe a system that presents a scrollable window of icons corresponding to significant points in a presentation that are determined by its author. An architectural design system developed by Makkuni [MAKK87] can simultaneously display separate copies of an editor interface for each of a series of actions performed by the user, visually linked to indicate their relationships.

In our work we have built on the visual metaphor of a comic strip—a sequence of *panels*, each of which illustrates an important moment in a story. The result is an *editable graphical history* that is generated automatically as the user interacts with a system, in our case a graphical editor. There are several important ways in which our system differs from earlier work:

Intra-panel content selection. Our system uses heuristics, similar in spirit to those of [FEIN85], to select the objects displayed in each panel and the style in which they are drawn in order to emphasize the objects being manipulated and the actions performed on them. In contrast, previous systems have used exact screen miniatures or user-specified icons.

Originally published in the *IEEE 1988 Workshop on Visual Languages*. Pittsburgh, PA. October 1988. pp. 127-134.
Reprinted in *Visual Programming Environments: Applications and Issues.* E.P. Glinert (ed.). IEEE Press, Los Alamitos, CA. 1990. pp. 416-423.

Inter-panel detail removal. The number of actions expressed in a single panel is determined automatically based on user preference and on a context-sensitive rating of the actions performed. Other systems show each action separately or show only author-designated actions.

Interactive elaboration. The user can request to see a given panel decomposed into lower-level ones or a set of panels coalesced into a higher-level one.

Full history editing. Our system allows the user to edit the actions expressed in the panels, deleting or modifying old actions and adding new ones. Previously-executed actions may be undone or modified, and subsequent actions redone. Unlike previous displays of undo possibilities [VITT84], graphical histories allow the user to view the past consequences of previous commands without having to execute an undo.

The following sections describe the visual language that we chose for the system and how it is implemented.

2. Basic Concepts

We believe that graphical interaction is best represented graphically. If the history mechanism uses the same visual language as the interface supported by this mechanism, then potential users of the histories need not learn a new language. As in GNU Emacs [STAL87], histories can be represented as lisp-expressions. Indeed, the histories in our implementation are represented internally as such. However, this representation is best hidden from the user if the history mechanism is to be used by programmers and non-programmers alike. In the case of graphical user-interaction, parameters to operations also tend to be entities (such as particular graphical objects or attributes) that are expressed relatively poorly in textual form, thus indicating a need for *graphical histories*.

We have been experimenting with graphical histories in the domain of graphical editing, although the ideas contained here can be applied to other graphical interfaces as well. In particular, we have constructed a history mechanism for a small graphical editor, called Chimera. All of the illustrations in this paper were created with the Chimera editor or by its graphical history mechanism. In both its user-interface and the types of interactive techniques that it supports, Chimera is patterned after the Gargoyle illustrator [BIER86, PIER88]. Chimera's history facility uses the same pictorial conventions as Chimera's user-interface, certain salient features of which will be explained in subsequent examples. In the remainder of this section we provide a high-level overview of Chimera's graphical history system, while postponing a detailed explanation to the following sections.

As a picture is edited in one window, pairs of panels that graphically describe the editing process are generated in another. The top panel of each pair, which we call *prologue*, shows the relevant portion of the scene prior to the fundamental operation or operations represented by the pair. Usually, the prologue contains those objects that can be construed as arguments to the operation. The bottom panel, the *epilogue*, depicts the part of the scene altered by these editing operations. Together, the pair represents "before" and "after" views of the portion of the scene acted on by a set of graphical editing commands.

Figure 1a displays the contents of a Chimera graphical editor window, and Figure 1b is a pair of history panels describing the editing sequence used to create Figure 1a. This figure, as well as those in the rest of the paper, is rendered from the PostScript [ADOB85] output of the Chimera editor, and its history mechanism. In Figure 1b, the first pair of panels depicts the sequence of line drawing operations that constructed the square at the top left of Figure 1a. The prologue shows only the caret (appearing as a Λ), which represents the editor's current position. Since one endpoint of a line is always based at the current caret location during the line drawing process, the caret acts as one argument to the line drawing command, and is important to include in the prologue of a panel pair representing one or more add line commands. The epilogue shows the square after it has been constructed. Note that this panel pair represents four add line commands.

The second, third, and fourth pairs of panels also show the construction of various shapes through add line commands. In the fifth pair of panels, we have selected two of these shapes to close (closed shapes are filled with the current fill color). The prologue shows those shapes that were selected to be closed, and neighboring shapes as well, and the epilogue displays these same shapes after the close operation was performed. Selected polylines are indicated by small, filled squares placed at the vertices. This is the same notation used by the editor itself, so users of the history facility are familiar with this. In the sixth pair we show a rotation executed on the unfilled triangle. The arguments to the rotation command include the anchor (drawn as a square with lines across its two diagonals), which acts as the center of rotation; the caret, which follows the hardware cursor during a rotation; and the selected objects, which are the objects acted upon by the rotation operation. Anchor, caret, and selected objects are represented visually in the history as they appear in the editor application. All of these arguments are included in the prologue of the rotation of command. The epilogue of the rotation command shows the relevant portions of the scene after the rotation has been performed.

Above each panel pair, we display the name of the main operation which it depicts, since this is not always immediately obvious from the before and after view. The operation names are those that the user is familiar with (through menus, documentation, and feedback echoed by the application). The history thus uses the same language used to generate the original picture. Also above each pair of panels is a pair of numbers that indicates how many editor operations were clustered automatically into the prologue and epilogue. For example, the first prologue contains a single operation, since the user made one caret-positioning command immediately prior to drawing the square. The epilogue contains eight editing commands, including the four line drawing commands and four implicit caret positioning commands called by the line drawing operations.

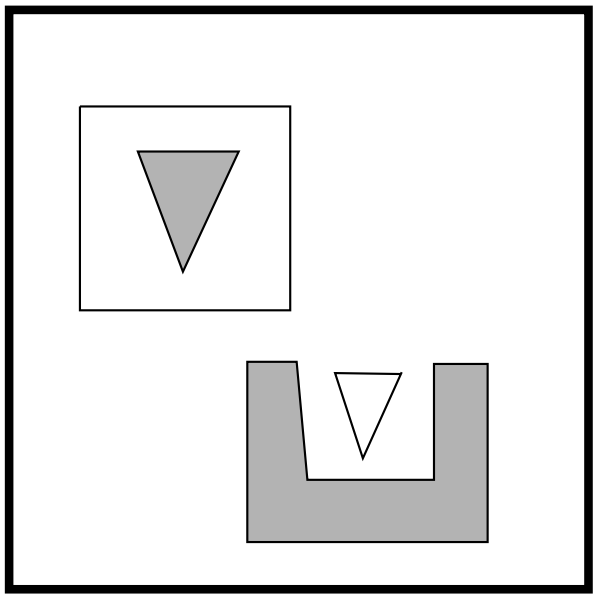
Any of these panel pairs can be expanded to show their component operations in more detail. If we expand the first panel pair of Figure 1b once, the individual line drawing operations are expanded into separate panels, as shown in Figure 2a. Similarly, in Figure 2b we expand the rotation represented by the sixth panel pair of Figure 1b into two pairs. The first of these portrays the placement of the anchor at the caret position. The second pair shows the rotation operation separated from the anchor placement.

3. Choosing Panel Contents

Since the size of each of the history panels is considerably smaller than the size of the edit window, it is usually necessary to restrict the objects displayed in the panels to a subset of those appearing in the entire picture. This is especially important for pictures that are more complex than the simple examples shown here. The scale of the objects appearing in the history panes and the portion of the scene shown is currently based on a number of heuristics similar to those used in the APEX system for generating pictorial explanations [FEIN85]. If the action represented by an epilogue acts upon the currently selected objects, these objects will be shown in the prologue as they appear before the operation, and in the epilogue as they appear afterwards. When an operation uses certain metaobjects as arguments, such as the caret or anchor, then these metaobjects will be drawn in the prologue at their positions before the operation sequence, and in the epilogue at their positions afterwards.

In the case that an object is in the process of being drawn, we include the rest of the object in both the prologue and epilogue. This provides a certain amount of context, which is necessary in order to help the user determine the spatial correspondence between the history panel and the scene in the editor at the time the operation was performed. The mechanism which chooses the panel contents always attempts to include in each a landmark—an object that can help to disambiguate the panel's relationship to the scene. The landmark should also be located extremely near (or ideally in) the portion of the scene that would otherwise be displayed in the panel. Window edges might also make reasonable landmark objects if the application does not have a scrollable canvas. This would be particularly useful with respect to the first panel of an editing sequence starting with an empty scene. Currently our mechanism for choosing landmark objects is primitive, and this part of the graphical history system has been targeted for future work.

Because Chimera uses the Snap-Dragging user-interface paradigm, described in [BIER86], it is usually clear when one object is being transformed or drawn directly to another. Objects have "gravity" and the caret, which is used both in transforming and drawing objects, is gravity sensitive. For example, when a collection of objects (including the caret) is translated, the collection has a propensity to snap into place in such a way so that the caret is attached to a nearby, immobile object. When objects are transformed or drawn directly to another object in the scene, this object also provides important context for the operation, and appears in the history panels. An interesting problem on which we are


(a)

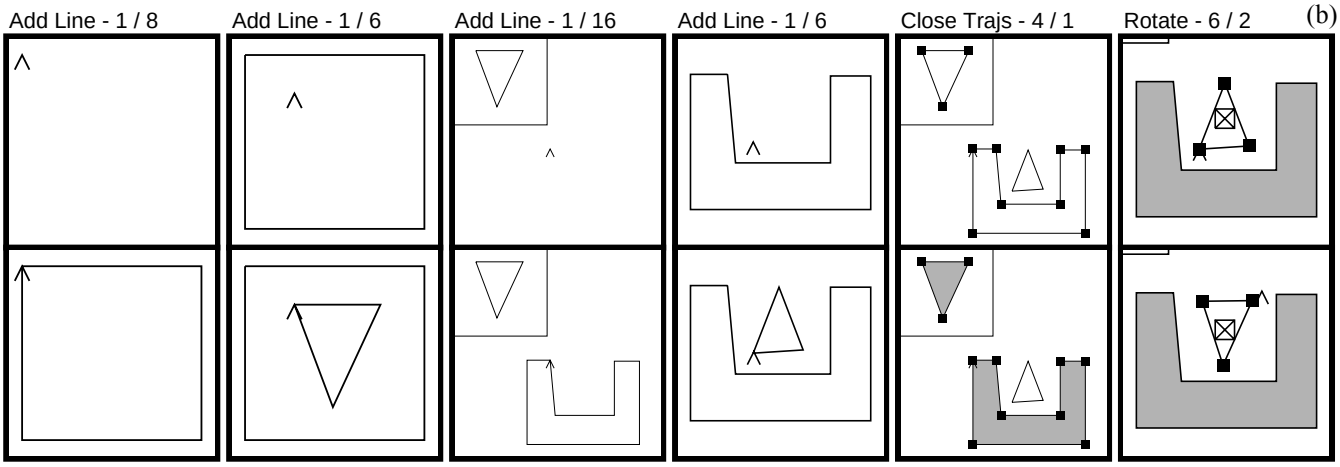


Figure 1. The Chimera editor. (a) The editor window. (b) Its graphical history.

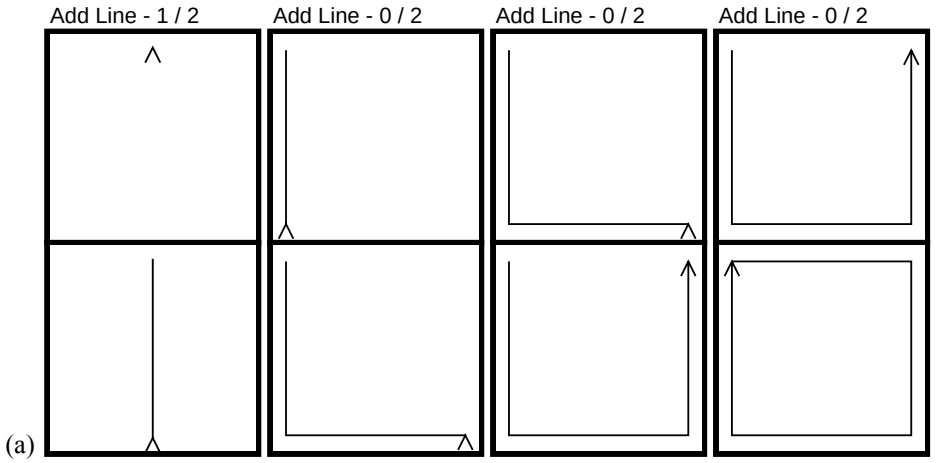


Figure 2. Expanded panels. (a) An expansion of the first panel pair of Figure 1b into its component Add Line commands. (b) An expansion of the last panel pair of Figure 1b into Place Anchor and Rotate commands.

Note: the total number of operations represented by the frames remains constant under expansion. Commands can migrate, however, between prologue and epilogue.

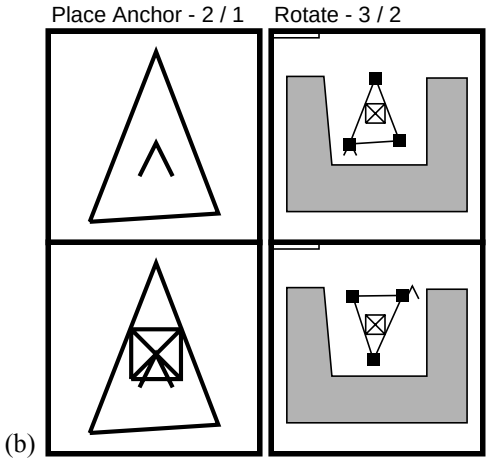
working involves determining how little of an object that is added to provide context can appear in a history panel, and still provide sufficient contextual information.

Finally, the system tries to have the prologue and epilogue map to the same region of the editing canvas, since these two panels are intended to provide a before and after view of a section of the scene. Occasionally this is not an optimal decision, as in the case where a small object is translated across the entire editor window. In this case, if the prologue and epilogue were both to include the initial and final position of the object, the scale of the panels might be such that the translated object were near invisible. If the merged area of the prologue and epilogue region extents is more than a preset constant times the sum of the individual extents, then separate regions are depicted in the prologue and epilogue. After two subregions of the editing window are chosen for display in the prologue and epilogue, the contents of these regions are isotropically scaled into the appropriate panels in the history window.

4. Panel Granularity

The number of commands that are represented in a single pair of panels is chosen automatically, based upon both a user-specified granularity level, and the sequence of operations that are actually performed. If the user is examining the edit history only to get a coarse understanding of what was accomplished in the session, he or she can specify a rough granularity level. A smaller granularity might be used by a person using edit histories for macro creation in order to have fine-grained control over operations. Users can also adjust the granularity level, based upon the amount of experience they have had with the system.

Certain sequences of commands are more readily coalesced into individual history panels than others. For example, sequences of translations of a single set of objects, without other intervening operations are represented by the same panel, since they are equivalent to a single translation. The same is done for rotations, scales, caret movements, and anchor placements. Sequences of consecutive add line commands, without any intervening caret movements, all relate to the drawing of a single polyline and are therefore



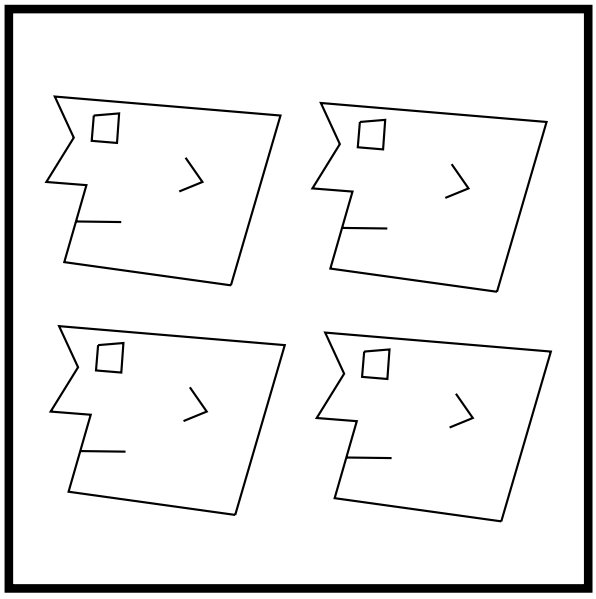
compressed into a single panel. Sets of object selections, deselections, caret movements, and anchor placements can frequently be interpreted as setting up the arguments for a subsequent operation. If this is so and the user-specified granularity level permits, all these operations are compressed into a single prologue history panel.

5. Using Histories for Undo and Redo

Editable graphical histories are useful in the specification of undo operations. The history is a timeline that represents the changing state of an application, and through the selection of a history panel, the user can ask for a previous state to be restored. The operations which are undone are saved away. At a later time, these operations can be replayed or discarded.

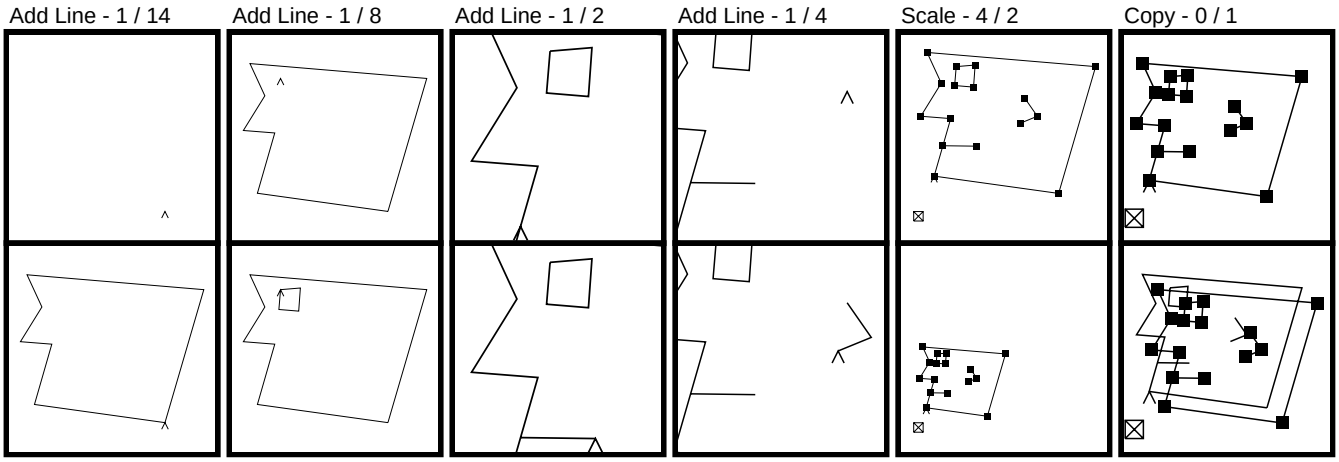
The following example illustrates the use of graphical histories for undo and redo in graphical editing. Figure 3a shows operations represented in the history panels of Figure 3b. Initially a single head was drawn, and this was in turn scaled down and copied. The copy was translated to the right of the original head. Both the original and the copy were selected and copied, and finally the new copy (of two heads) was translated above the other two. We would like to close and fill the trajectories (polylines) that represent the eyes of the

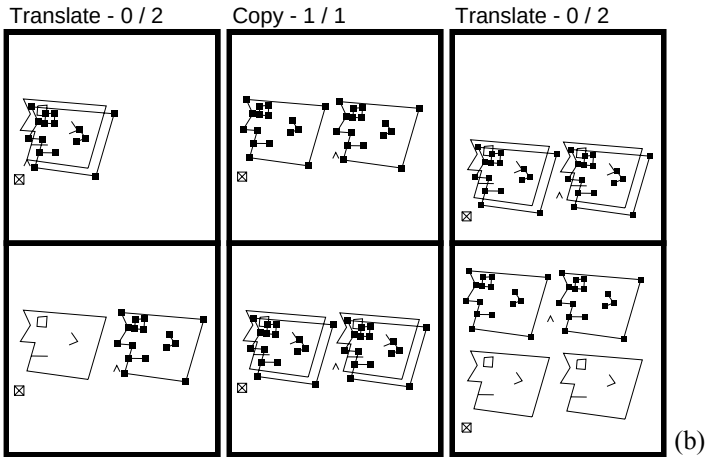
heads. Although we could select each of the eye trajectories, and invoke the close command without using graphical histories at all, this could be tedious, especially if the number of copies were large, and would require deselecting other currently selected objects.



(a)

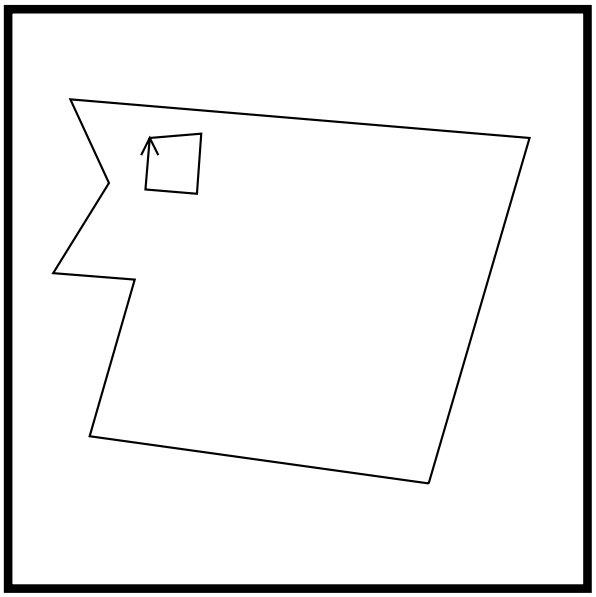
Initially the artist restores the editor state to the scene immediately after the eye was originally drawn but before any copies were made, by selecting the epilogue of the second panel pair in Figure 3b. The resulting editor window appears in Figure 4a. Next the eye trajectory is selected and closed, automatically causing the eye to be filled with the current fill color (gray). The editor window after this step is shown in Figure 4b. Finally, we execute a single command which automatically redoes the operations that we had undone, copying the head multiple times and placing the copies as before. Figure 5a shows the illustration generated by this process: four heads with gray eyes. The graphical history updates itself throughout this process, and at the end of this editing task, when Figure 5a been generated in the editor window, the graphical history in Figure 5b appears in the edit history window. Note that this edit history is similar to that displayed before the undo and redo, but it now includes a new Close Trajs panel pair, which records the close operation on the eye trajectory. All of the subsequent panels contain closed and filled rectangles for the eyes.



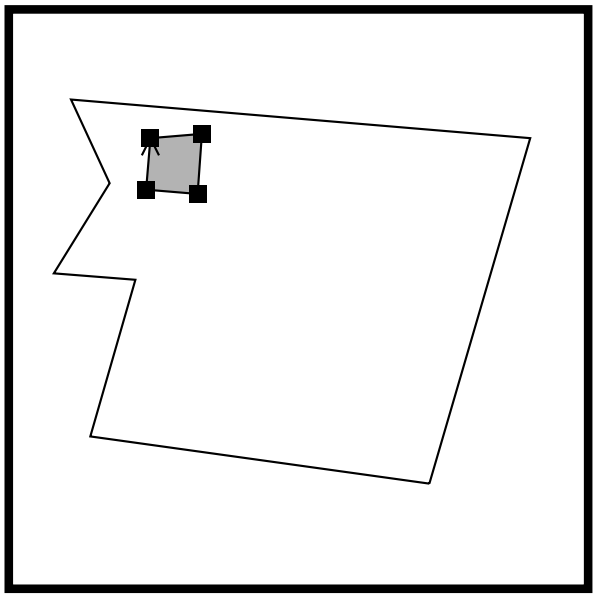


(b)

Figure 3. Four heads. (a) The editor window. (b) Its graphical history.



(a)



(b)

Figure 4. Undo. (a) The editor window, loaded with the epilogue of Figure 3b, panel pair 2. (b) The editor window after editing.

6. Implementation

The Chimera graphical editor and its graphical history mechanism were implemented in Common LISP on HP 9000/300 series workstations. Chimera runs under the NeWS window system, and communicates to NeWS through a set of C and PostScript routines.

As the user edits a picture, Chimera builds up an internal edit history containing key LISP expressions which were called to construct the picture. These expressions can be re-evaluated to rebuild the illustration from a check-pointed editor scene structure. These expressions are also interpreted in the context of the scene data to build the graphical history. The first step in building the graphical history involves clustering together sets of history commands into groups that can be represented by a single panel. In determining whether to add another command to a panel, the clustering mechanism takes into account the new command, the commands that have already been clustered in the panel, whether the panel is a prologue or an epilogue, and a user-chosen granularity level. Though in some cases we might be able to compose better history panels knowing what commands the user will execute later, lookahead is not used in the process of creating history panels. This allows history frames to be built concurrently with the editing process, without constantly re-evaluating the clustering of already constructed panels.

In the next step, the history mechanism chooses the portion of the editor window to map to the panel being constructed. Each editor operation that can be invoked by the user has a corresponding function that is called when the operation is clustered in the panel being constructed. This function examines the scene data-structures to choose a region of relevance to its operation. As discussed earlier in Section 3, this region may include all selected objects, the anchor, the caret, an object in the process of being drawn, or other entities, depending upon the operation. After suggestions have been gathered for all operations to be represented by the panel, the suggestions are combined with a few other heuristics (e.g., include a landmark in the panel, and have prologue and epilogue display the same region of the editor window if possible) in order to choose a region to be depicted by the panel. The display of the prologue is postponed until after its epilogue has been processed to allow the system to show the same region in both.

7. Conclusions and Future Work

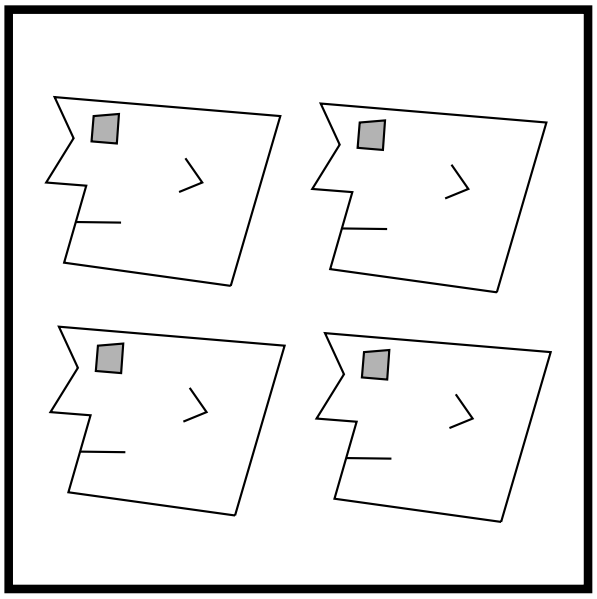
Graphical histories add to a user-interface a visual record of past events. We represent this record as a set of panels, containing a pair for each significant user-interaction event. This visual record uses the same visual metaphors as the interface, thus making it accessible to anyone familiar with the rest of the system. Using this timeline as a guide, a person can select a past system state to be restored, thereby supporting the one-step specification of an undo operation. After an undo operation, new operations can be performed, followed optionally by an automatic redo of all undone operations.

Our graphical history system differs from earlier work on user-interface timelines in that it automatically determines how many operations to coalesce into a single history panel. The decision is based upon how well the commands being considered are known to group together, and a user-specified granularity level. Unlike previous systems, ours also builds its own history icons, based upon the operations being performed and the system state. These customized panels can better convey the effect of the user's commands given the current system state than prefabricated icons or miniatures of the entire interface. Since panels can represent more than one operation, they can be split into lower level panels that show their constituent operations in more detail.

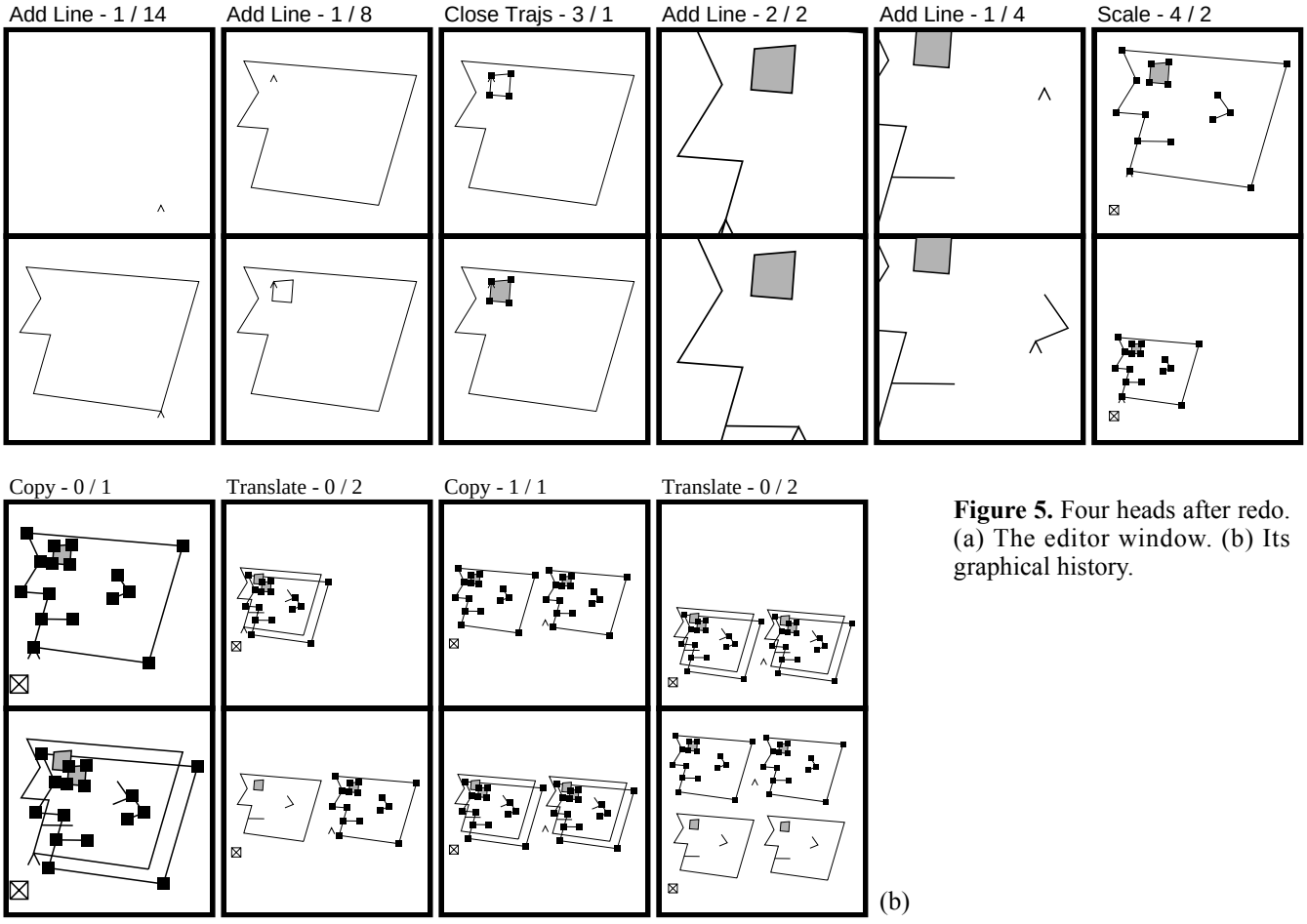
Similarly, multiple panels can be joined into a single panel to hide this detail.

We are in the process of extending the mechanism that determines how to split up panels, and how much detail to place in each. Currently our system removes detail by limiting the view shown in history panels to be only the portion of the graphical display containing relevant information. It would also be reasonable at times to remove objects from the panels that fall in the interesting regions of the display, but are not relevant themselves. The problem of choosing good landmarks is an interesting one, and deserves more attention in our system. We are also working on an interface which will permit the splitting and rejoining of history panels to be viewed in such a way that the original panel hierarchy is always clear.

A mechanism to facilitate the fast recall of old history panels is certainly desirable. Implementing "fisheye views" [FURN86], modulating information detail as a function of conceptual proximity, will help with this task. More recent events would appear in detail, distributed among a large number of history panels. Operations performed longer ago would be more densely packed in history panes (which can be expanded at will). It would also be useful to permit the retrieval of old history panels quickly through a specification of the objects affected or operations of interest, e.g., through graphical search techniques [KURL88]. The ability to ask for a history of a set of one or more particular objects



(a)



(b)

Figure 5. Four heads after redo. (a) The editor window. (b) Its graphical history.

would be a helpful feature to have in the generation of history presentations.

In graphical editor histories, it is sometimes desirable to show panels that contain a view taken from outside the graphical editor scene. When the user changes a button setting, slider value, or text-input region in an editor control-panel, the change should be included in the graphical history as well. Extensions, such as these, that we plan to make to our graphical history mechanism will be important in applying editable graphical histories to more general user-interfaces.

The undo mechanism described in this paper is referred to as linear undo/redo in the classification scheme described by Vitter [VITT84]. We hope to extend this mechanism so that more than one set of possible redo operations are available at any given time. Graphical histories would mesh well with the more robust redo facility, since a traditional problem with multiple redo options is in the presentation of the redo choices to the user.

Graphical macros are of special interest to us, and this work suggests one possible approach to representing them visually. A suitable visual presentation of macros is important during both macro construction, and editing. Graphical histories would also be useful in selecting past actions to be encapsulated in macros, and in reviewing macros which have been archived in a library. We are actively pursuing research in this area.

References

[ADOB85] Adobe Systems Inc. *PostScript Language Reference Manual*. MA: Addison-Wesley, 1985.

[BIER86] Bier, E., and Stone, M. "Snap-Dragging." *Proc. SIGGRAPH '86.* In *Computer Graphics*, 20:4, August 1986, 233-240.

[CHRI88] Christodoulakis, S., and Graham, S. "Browsing Within Time-Driven Multimedia Documents." *Proc. Conf. on Office Info. Sys.*, March 23-25, 1988, 219-227.

[FEIN82] Feiner, S., Nagy, S., and van Dam, A. "An Experimental System for Creating and Presenting Interactive Graphical Documents." *ACM Trans. on Graphics*, 1:1 January 1982, 59-77.

[FEIN85] Feiner, S. "APEX: An Experiment in the Automated Creation of Pictorial Explanations." *IEEE Computer Graphics and Applications*, 5:11, November 1985, 29-38.

[FURN86] Furnas, G. "Generalized Fisheye Views." *Proc. CHI '86 Human Factors in Computing Systems*, Boston, April 13-17, 1986, 16-23.

[JOY79] Joy, W. An Introduction to the C Shell. In *UNIX Programmer's Manual, Seventh Edition, Third Berkeley UNIX Distribution*, Dept. of EE & CS, University of California, Berkeley, 1979.

[KURL88] Kurlander, D., and Bier, E. "Graphical Search and Replace." *Proc. SIGGRAPH '88.* In *Computer Graphics*, 22:4, August 1988, 113-120.

[MAKK87] Makkuni, R. "A Gestural Representation of the Process of Composing Chinese Temples." *IEEE Comp. Graphics and Applic.*, 7:12, December 1987, 45-61.

[PIER88] Pier, K., Bier, E., and Stone, M. "An Introduction to Gargoyle: An Interactive Illustration Tool." *Proc. EP88, Int. Conf. on Electronic Publishing, Document Manipulation, and Typography*, April 1988, 223-238.

[SMIT78] Smith, A.R. "Paint." NYIT Computer Graphics Lab Technical Memo No. 7, Old Westbury, NY, July 20, 1978. [Also available in Beatty, J. and Booth, K. (eds.), *IEEE Tutorial: Computer Graphics 2nd Ed.* Silver Spring, MD: IEEE Comp. Soc. Press, 1982, 501-515.]

[STAL87] Stallman, R. *GNU Emacs Manual.* Sixth Edition, Version 18, Cambridge, MA, Free Software Foundation, March 1987.

[SWIN86] Swinehart, D., Zellweger, P., Beach, R., and Hagmann, R. "A Structural View of the Cedar Programming Environment." *ACM Trans. on Progr. Lang. and Sys.*, 8:4, 419-490, 1986.

[VITT84] Vitter, J. "US&R: A New Framework for Redoing." *IEEE Software*, 1:4, October 1984, 39-52.