# A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands

David Kurlander
Steven Feiner

Department of Computer Science
450 Computer Science Building
Columbia University
New York, NY 10027

djk@cs.columbia.edu
feiner@cs.columbia.edu

*February 1989*

## Abstract

We present the concept of an *editable graphical history* that allows the user to review and modify the actions performed with a graphical user interface. Using a pictorial metaphor borrowed from comic strips, an editable graphical history consists of a series of panels that depict in chronological order the important events in the history of a user's session. We discuss the visual language used in editable graphical histories, and describe Chimera, a graphical editor that generates these histories automatically. The user may scroll through the sequence of panels, reviewing actions at different levels of detail, and selectively undoing, modifying, and redoing previous actions.

Chimera's editable graphical histories are constructed from parts of the editor window, the editor control panel, and the editor's pop up menus. Panels indicate both the objects that are modified and the actions performed on them. We describe the heuristics used to determine the objects depicted in each panel, the style in which they are drawn, and how actions are distributed among panels.

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

**Reprinted from *Visual Languages and Visual Programming,* S. K. Chang (ed.). Plenum Press, New York, NY. pp. 257-275. 1990.**

# A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands

David Kurlander
Steven Feiner

## *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*,[(7)] allow the user to list, edit, and reexecute previously executed commands. Many window-based systems further provide the ability to scroll over the entire transcript of a session and select text for reexecution.[(13)] 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.

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 reexecution*, 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 long been an important feature in graphical interfaces such as those of computer paint systems.[(11)] 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.[(12)] These techniques are most commonly used for undoing or modifying past actions, rather than merely browsing.

There have been several attempts to merge the spatial browing and temporal reexecution history paradigms for graphical interfaces. Feiner, Nagy, and van Dam's IGD hypermedia system[(4)] allows readers of its graphical documents to view an automatically constructed timeline presenting 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[(3)] 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[(9)] can simultaneously display separate copies of an editor interface for each of a series of user actions, 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. The simplest approach to generating an editable graphical history would be to map each operation to a panel and use an exact miniature of the screen for each panel. This was the approach taken in IGD's timelines, in which a new miniature was added to the timeline for each page visited. It has two major problems. First, mapping each operation to its own panel results in a proliferation of panels, especially when operations are performed rapidly by experienced users. Second, mapping the entire screen to each panel means that successive panels with small differences that result from minor changes will be difficult to tell apart. In addition, since panels are typically much smaller than the workspace being depicted, the panel contents for a real application may be nearly illegible. Unfortunately, the worst cases may be those in which the history mechanism is needed most.

Recognizing these problems, we have provided several important features:

*Inter-panel detail removal.* Our approach addresses the problem of one-to-one mapping of actions to panels by coalescing multiple actions into a single panel by performing a context sensitive evaluation of the actions. The number of actions expressed in a single panel is determined automatically based on user-specifiable granularity level. Previous systems show each action separately or show only author-designated actions.

*Intra-panel content selection.* We address the problem of one-to-one mapping of the screen to a panel by using heuristics, similar in spirit to those of APEX,[5] 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.

*Interactive elaboration.* Since it is often useful to view certain parts of the history at a different level of detail from other parts, we allow the user to request that a given panel be decomposed into lower-level ones or that a set of panels be coalesced into a higher-level one.

*Full history editing.* Our system allows the user to browse the history, undoing, modifying, and redoing previous actions. Unlike previous displays of undo possibilities, however, our graphical histories allow the user to view the past consequences of previous commands without having to execute undos or redos.

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,[12] 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. Furthermore, the user should be able not only to review the past state, but also to change it, thus suggesting that graphical histories be *editable*.

If sufficient processing power is available, one approach to providing editable graphical histories would be to use pure temporal reexecution, allowing the user to scroll the main window smoothly back and forth in time, and to edit it at any point. Although this approach might work well for a small history, it does not generalize to histories of significant length and complexity. For example, small sequential changes in vastly differing parts of the display would be difficult to recognize. By itself, smooth temporal scrolling provides no means of determining and indicating the session's highlights, of removing both spatial and temporal detail, or of focusing the viewer's attention on the relevant parts of the workspace. In essence this would be like reviewing a long, linear scroll of text that had no chapter or section headings. Although smooth temporal scrolling is a useful technique, we believe that it should supplement, not replace, the techniques that we describe here.

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 graphical editor, called Chimera. All of the illustrations in this paper were created with the Chimera editor and 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.[2, 10] 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 the *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 fundamental operations. The bottom panel, the *epilogue*, depicts the part of the scene altered by these editing operations. Together, the pair represent 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 set of history panels from Chimera's history window 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[1] 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 concave quadrilateral at the bottom 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 it in the prologue of a panel pair representing one or more Addline commands. The epilogue shows the quadrilateral after it has been constructed. Note that this panel pair represents four Addline commands.

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 views. The operation names are those with which the user is familiar (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 prologue of the first panel pair contains a single operation, since the user made one caret-positioning command immediately prior to drawing the quadrilateral. The epilogue contains four editing commands, one for each line drawn. (The epilogue actually contains an additional four implicit caret positioning commands that are called by the line drawing operations, one for each line drawn. These implicit commands are suppressed from the command count, however, to avoid confusing the user.)

The prologue of the second panel pair shows the selection of the quadrilateral. 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 already familiar with it. The user has prepared to set the strokewidth of the selected lines by entering a value into the text input box of the editor's control panel. Because the control panel is used in the operation, Chimera includes both it and the selected objects in the prologue. The text input box is included at the top of the prologue. The epilogue shows the thicker quadrilateral that results from having chosen Setstrokewidth from the menu.

The third panel pair show the addition of a filled rectangle. Note that the previously drawn quadrilateral is now rendered in a subdued style, indicating that it does not play a part in the action being depicted. In fact, this quadrilateral has been included in this set of panels to act as a *landmark* that might help in identifying the location of the rectangle being added. Currently, the user is allowed one of several rendering styles for these less important objects, including invisible, subdued, and regular. The subdued style shown here is implemented by desaturating the color of the object being rendered.

The fourth and fifth panel pairs show the addition of a circle and some text. In the sixth panel pair we see the text being scaled. The arguments to the scale command include the *anchor* (drawn as a square with lines across its two diagonals), which acts as the point about which scaling occurs; the caret, which follows the hardware cursor during a scaling operation; and the selected objects, which are the objects acted upon by the scaling 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 scaling command. The epilogue of the scaling command shows the relevant portions of the scene after scaling has been performed.

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. Each epilogue shows a single line being drawn. The first pair's prologue shows a single caret positioning command, but each of the three remaining prologues shows no commands being executed. Similarly, in Figure 2b we expand the scaling operation 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 scaling operation separated from the anchor placement. The caret and anchor are shown in their regular style in both panel pairs because the caret acts as an argument to the Placeanchor command in the first pair, and both the caret and anchor participate in the scaling operation, along with the text, in the second pair. Note that the total number of operations represented by the frames remains constant under expansion. Commands can migrate, however, between prologue and epilogue.

## *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 panels 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.[5] 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 that chooses the panel contents always includes a landmark in each, if possible. A landmark is an object that helps 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,[(2)] 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 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 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 maps the prologue and epilogue to the same region of the editing canvas, unless this results in making the interesting objects in either of the two panels too small. Mapping the prologue and epilogue to identical regions usually provides for an easier before-and-after comparison. In some instances, however, showing the same region in the prologue and epilogue will actually obscure the action being performed. For example, when a small object is translated across the entire editor window, if the prologue and epilogue were both to include the initial and final positions of the object, the scale of the panels might be such that the translated object would be nearly 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 scaled isotropically into the appropriate panels of 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 is 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 finer granularity might be chosen by a person using edit histories for macro creation in order to have precise 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 Addline commands, without any intervening caret movements, all relate to the drawing of a single polyline and are therefore coalesced 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 coalesced into a single prologue history panel. The portion of Chimera's history mechanism that determines how to coalesce commands is rule-based. Sets of rules, predicated on granularity level, determine which commands can be placed together in the prologue, and which in the epilogue, of a history panel pair.

Figure 3 demonstrates how Chimera collapses panel pairs as they are built. In the first three panel pairs of Figure 3a the user selects the word ''Visual'', places the anchor, and changes the font of the selected text. In the next three panel pairs, the circle is also selected, the anchor moved to a new location, and the caret moved as well. If the user next performs a rotation, then the rotation is collapsed along with the last three panel pairs into a single panel, shown at the end of Figure 3b. Note that the Topselect and Placeanchor operations in the first and second panel pairs of Figure 3a are similar to those of the fourth and fifth panel pairs of the same figure. The final sequence of Topselect, Placeanchor, Movecaret, and Rotate is collapsed because it matches a coalescing rule for Rotate. In contrast, the earlier sequence of Topselect and Placeanchor is not coalesced since it is not matched by a coalescing rule for any of the following actions. It would have been coalesced at the current granularity, however, if followed by a Rotate or any other command that uses the anchor and selected objects.

## *5. Using Histories for Undo and Redo*

Editable graphical histories are useful for undoing and redoing commands. The history is a timeline that represents visible changes in an application's state. Through the selection of a history panel, the user can ask for a previous state to be restored. The operations that are undone are then saved away, and at a later time, can be redone or discarded.

Edit operations performed after an undo can sometimes cause a subsequent redo to fail. For example, if operations referring to a particular scene object are undone, and then the scene object is deleted, the undone operations cannot be redone. When this is attempted, Chimera detects the inconsistency and prints out an appropriate message.

The following example illustrates the use of graphical histories for undo and redo in graphical editing. Figure 4a shows a drawing of a bar chart, which was produced by the sequence of operations represented in the history panels of Figure 4b. The history panels shown here begin after the chart's frame and text have been drawn. First a single bar is drawn. Then it is copied (the copy is automatically offset below and to the right) and dragged into position to the right of the original bar. Since the original bar was deselected when the copy was made, it is reselected, and the pair of selected bars is copied and dragged to the right, to create a total of four bars. One at a time, the top edge of each of the last three bars is individually selected and dragged upwards to make the bar taller. (Note that the panel focusing on the penultimate bar makes the bar look thinner than in other panels. This is because this panel must map to a larger portion of the editor scene to contain this bar in its entirety.) Finally, the chart's border is selected and its strokewidth increased.

At this point we decide that we would like to increase the strokewidth of the bars and change their fillcolor. Although we could select each of the bars and apply the change to them, this could be especially tedious if there were a large number, and would also require deselecting other currently selected objects. Using the graphical history, however, we can restore the editor state to the time immediately after the creation and selection of the first bar, but before any copies were made, by selecting the prologue of the second panel pair in Figure 4b. Chimera then loads the editor window from this panel, as shown in Figure 5a. Next the bar's strokewidth and fillcolor are changed using the control panel's text input box twice, the second time specifying an rgb color. The editor window after this step is shown in Figure 5b. Finally, we execute a single command that automatically redoes the operations that we had undone, copying the bar multiple times, placing the copies as before, and changing the strokewidth of the chart's border. Figure 6a shows the chart generated by this process, which contains four lighter bars with thicker strokewidths. The graphical history updates itself throughout this process, and as Figure 6a is generated in the editor window, the graphical history in Figure 6b is generated in the edit history window. Note that this edit history is similar to that displayed before the undo and redo, but it now includes new Setstrokewidth and Setfillcolor panel pairs that record the newly added operations.

## *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. Currently Chimera is in use by the authors for general graphical editing, and we plan to release it to our local user community after enhancing the user interface.

As the user edits a picture, Chimera builds up an internal edit history containing key LISP expressions that were called to construct the picture. These expressions can be reevaluated to rebuild the illustration from a check-pointed editor scene structure. For each operation in the edit history, Chimera also maintains an undo expression. This permits efficient reconstruction of past states. The history expressions are interpreted in the context of the scene data to build the graphical history. When a graphical editor operation is performed, history expressions are constructed, and passed along to the mechanism that generates the graphical history.

Each editor operation has associated with it a set of rules for coalescing it with other operations preceding it on the edit history stack. The firing of these rules is predicated on the granularity level set by the user, thus allowing the coarseness of the presentation to be customized. The coalescing rules also determine which operations are to be associated with the prologue and the epilogue. If any of these operations had been previously associated with another history panel pair, that panel pair is discarded, and a new pair constructed.

For example, here are the rules associated with the Rotate command:

*coarse*; (Select-op | Anchor-op | Movecaret)$^*$; (Rotate)$^+$
*medium*; (Select-op | Anchor-op | Movecaret)$^*$; Rotate

*fine*; (Select-op | Movecaret)$^*$; Rotate

*finest*; ( ); Rotate

These rules are tested in order, and only the rule that matches first is used. The first element of every rule is a granularity level below which the rule will not fire. The second element matches a sequence of operations for the prologue, and the third element matches a sequence of operations for the epilogue. In the rules above we use regular expressions to match on these sequences, but arbitrary functions can also be used. The prologue and epilogue expressions must together match a non-zero length sequence of commands executed by the editor.

In this discussion, we refer to *coarse* granularity as the greatest granularity level, and *finest* as the least. As declared in the above rules, at the *coarse* granularity level a sequence of one more Rotate commands will be coaelesced in the epilogue, and all of the selection operations, anchor placement operations, and caret motion operations immediately preceding the Rotates will be placed in the prologue. At the *medium* granularity level and below, we will only permit a single rotate operation in the epilogue. At the *fine* granularity level and below, anchor placement operations will not be allowed in the prologue, and at the *finest* granularity, Rotation commands are represented in panel pairs by themselves. Note that at all of these granularity levels, Rotate commands can appear in panels by themselves if there are no preceding operations matched by prologue and epilogue expressions. In Figure 3, a Rotate command collapsed prior panel pairs so that selection, anchor placement, and Movecaret operations were redistributed to the prologue of a new panel pair. According to the rules above, this will happen at the *medium* granularity level and above.

After the coalescing rules have been applied, they will have constructed a panel pair representing a single logical operation, although possibly composed of several real operations. With each editor command, we associate a function that specifies which objects should be displayed in the prologue and epilogue. The functions also label these objects with one or more tags that indicate the reason they have been added to the panel, in the manner of APEX.[5] These tags currently include *landmark*, *selected object*, *metaobject*, *operation argument*, *context*, and several different relations that an object can have with the caret (e.g., an object to which the caret has been attracted).

Again, choosing the Rotate command as our example, we execute a function that includes the caret, anchor and selected objects in both the prologue and epilogue, since these are important in understanding both the specification and effect of this operation. The caret and anchor are marked as metaobjects and operation arguments. Object to be included in the history panels, but that are not part of the scene, are tagged as metaobjects; objects that act as arguments to the the editor operation being performed are tagged as operation arguments. The selected scene objects are tagged as selected, and also as operation arguments, since they too act as arguments to the Rotate command. A landmark is found for each panel, and the landmark is tagged. Tags are later referenced by the renderer, which uses this information in selecting a rendering style for each object.

After determining which objects must be included in the panels, other heuristics are used to choose appropriate bounding boxes of the regions to be depicted. For example, these guidelines include having the prologue and epilogue display the same regions of the editor window if this will not diminish the size of the important objects too much, and adding margins so that the edges of important objects do not coincide with the edge of the panel. When the bounding box is set, a final pass is made to identify those objects that fall within the visible region, but are not of special interest themselves. These objects are tagged as *background*, and can provide important contextual cues to the viewer.

Finally, the renderer is called to draw all objects in each bounding box. They are rendered in a style that depends on how they have been tagged. The rendering style can be customized easily by the user, accounting for individual preferences and differences in the capabilities of the rendering hardware. Currently, Chimera allows contextual objects (landmarks and background objects) to be rendered normally, invisibly, with desaturated colors, or with brightened colors.

## *7. Conclusions and Future Work*

Editable graphical histories add to a user interface a visual record of past events. We represent this record as a set of panels, containing a before-and-after 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 the graphical history as a guide, a person can select a past system state to be restored, thereby supporting the one-step specification of a multi-level 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 a context-sensitive determination of how well the commands being considered group together, and a user-specified granularity level. Unlike previous systems, ours also designs its own history panels, 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.

Currently our system removes detail by limiting the view shown in history panels to be only the portions of the editor window and interaction device windows that contain relevant information. The ability to display interactions occurring on multiple parts of the display is an important first step toward applying editable graphical histories to more general user interfaces.

Several rendering styles are made available to the user to determine how to depict the objects that fall within these interesting areas of the display, but which are not directly relevant to the actions. The effectiveness of each of these styles depends in part on the picture for which it is used. Ideally, the system itself should determine the most appropriate rendering approach.

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 will not split up a single action among several panels, though in some cases this would be useful. For example, if many small objects, located far apart, are changed with a single operation, then they may appear too small in the history panels for the change to be noticable. The editor should be capable of expanding this panel pair, even though it represents a single operation, to show detailed views of all the objects that have been modified.

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’’[(6)] that modulate 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 panels (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.[(8)] The ability to ask for a history of a set of one or more particular objects would be a helpful feature to have in the generation of history presentations.

Chimera’s user interface currently supports only what Vitter terms linear undo/redo.[(14)] We intend to extend the interface so that multiple sets of possible redo operations are made available at any given time. Graphical histories would mesh well with this richer redo facility, since a traditional problem with multiple redo options is how to present 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 that have been archived in a library. We are actively pursuing research in this area.

*Acknowledgements*

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.

*References*


1. Adobe Systems Inc. *PostScript Language Reference Manual*. Reading, MA: Addison-Wesley, 1985.
2. Bier, E., and Stone, M. ‘‘Snap-Dragging.’’ *Proc. SIGGRAPH ’86* (Dallas, TX, August 18-22, 1986). In *Computer Graphics*, 20:4, August 1986, 233-240.
3. Christodoulakis, S., and Graham, S. ‘‘Browsing Within Time-Driven Multimedia Documents.’’ *Proc. COIS88, Conf. on Office Info. Sys.*, Palo Alto, CA, March 23-25, 1988, 219-227.
4. 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.
5. Feiner, S. ‘‘APEX: An Experiment in the Automated Creation of Pictorial Explanations.’’ *IEEE Comp. Graphics and Applic.*, 5:11, November 1985, 29-38.
6. Furnas, G. ‘‘Generalized Fisheye Views.’’ *Proc. CHI ’86, Human Factors in Computing Systems*, Boston, MA, April 13-17, 1986, 16-23.
7. 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.
8. Kurlander, D. and Bier, E. ‘‘Graphical Search and Replace.’’ *Proc. SIGGRAPH ’88* (Atlanta, GA, August 1-5, 1988). In *Computer Graphics*, 22:4, August 1988, 113-120.
9. Makkuni, R. ‘‘A Gestural Representation of the Process of Composing Chinese Temples.’’ *IEEE Comp. Graphics and Applic.*, 7:12, December 1987, 45-61.
10. 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*, Nice, France, April 1988, 223-238.
11. 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.]
12. Stallman, R. *GNU Emacs Manual.* Sixth Edition, Version 18, Cambridge, MA: Free Software Foundation, March 1987.
13. Swinehart, D., Zellweger, P., Beach, R., and Hagmann, R. ‘‘A Structural View of the Cedar Programming Environment.’’ *ACM Trans. on Prog. Lang. and Sys.*, 8:4, October 1986, 419-490.
14. Vitter, J. ‘‘US&R: A New Framework for Redoing.’’ *IEEE Software*, 1:4, October 1984, 39-52.

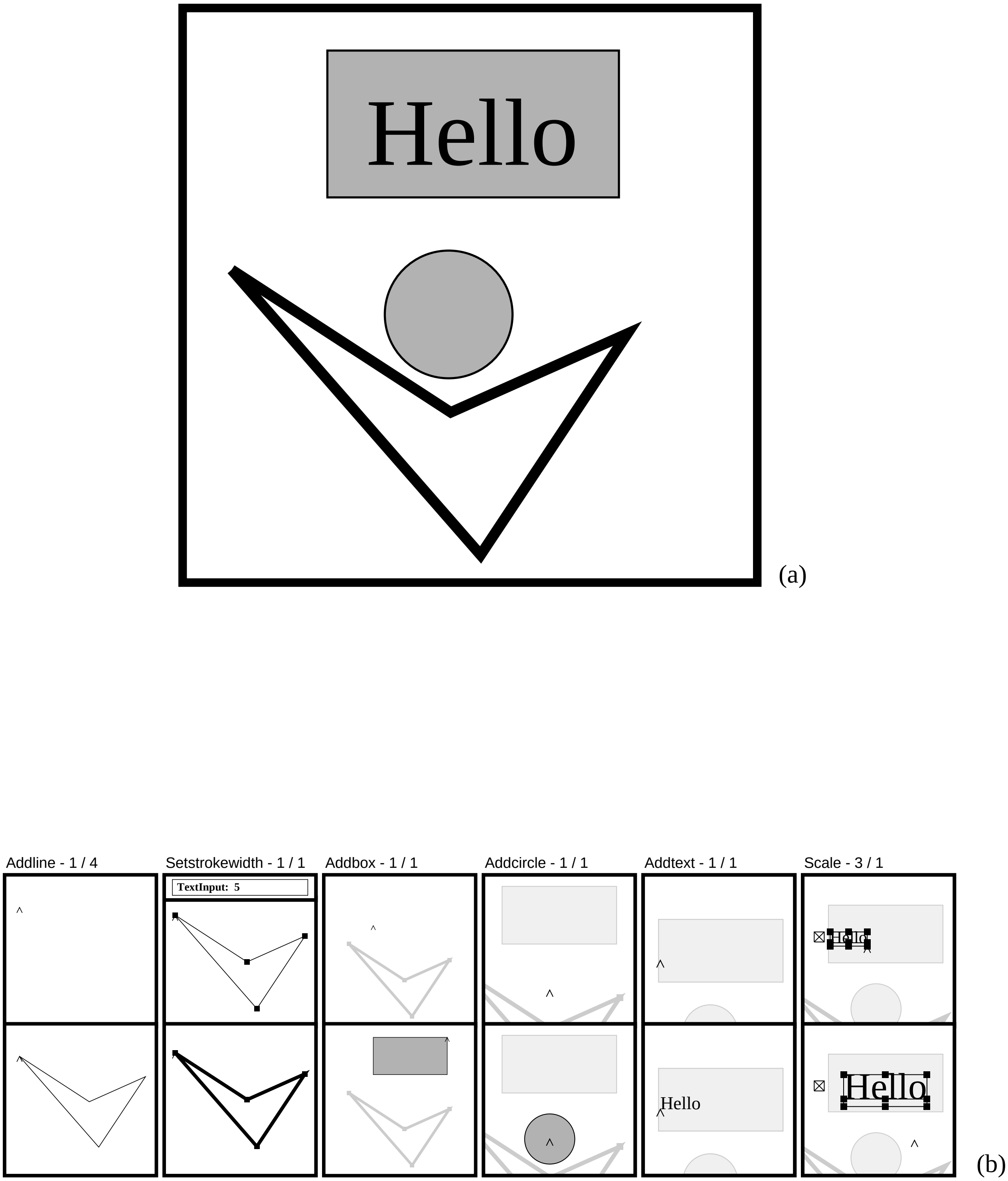
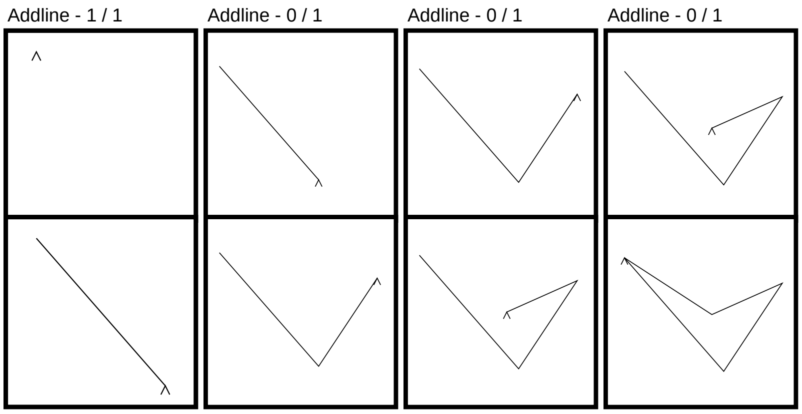
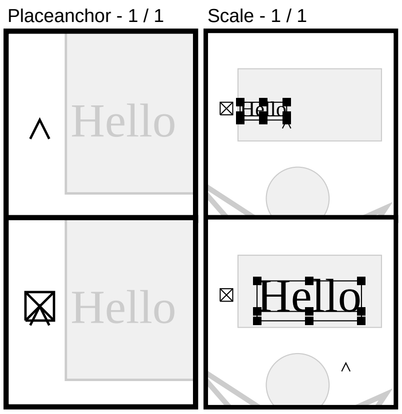



Figure 1. The Chimera editor. (a) The editor window. (b) Its graphical history.

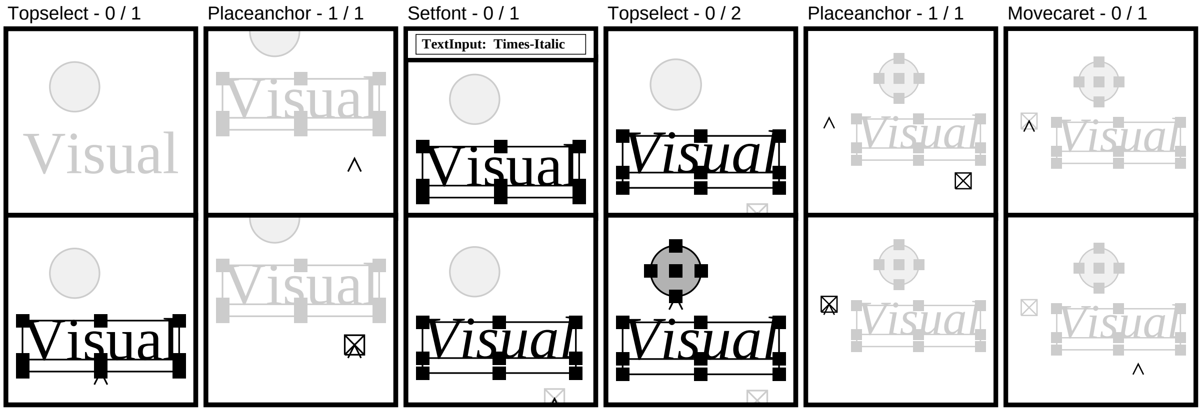



(a)

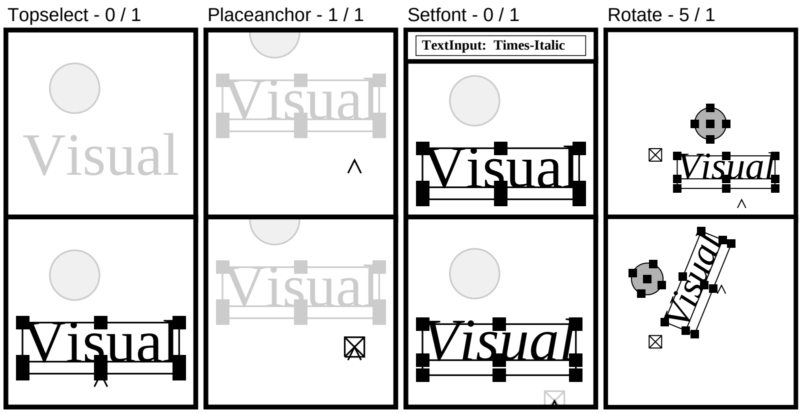



(b)

Figure 2. Expanded panels. (a) An expansion of the first panel pair of Figure 1b into its component Addline commands. (b) An expansion of the last panel pair of Figure 1b into Placeanchor and Scale commands.

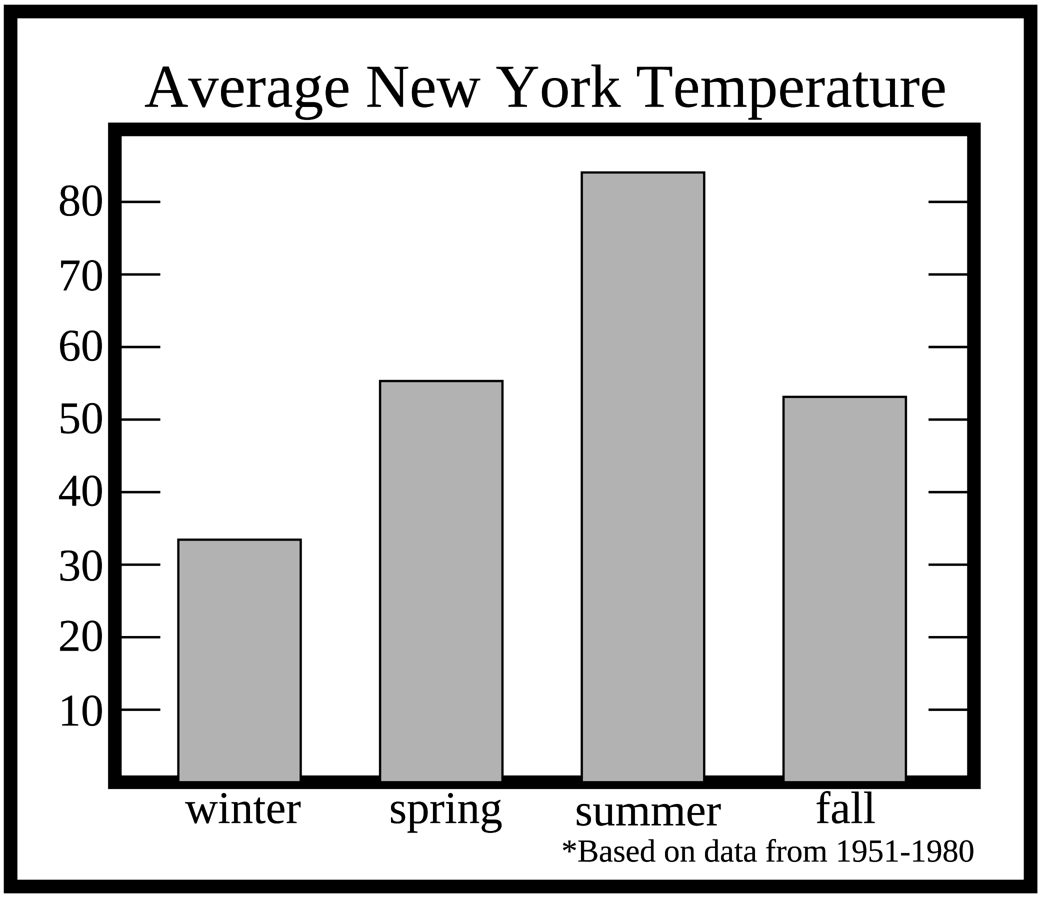



(a)

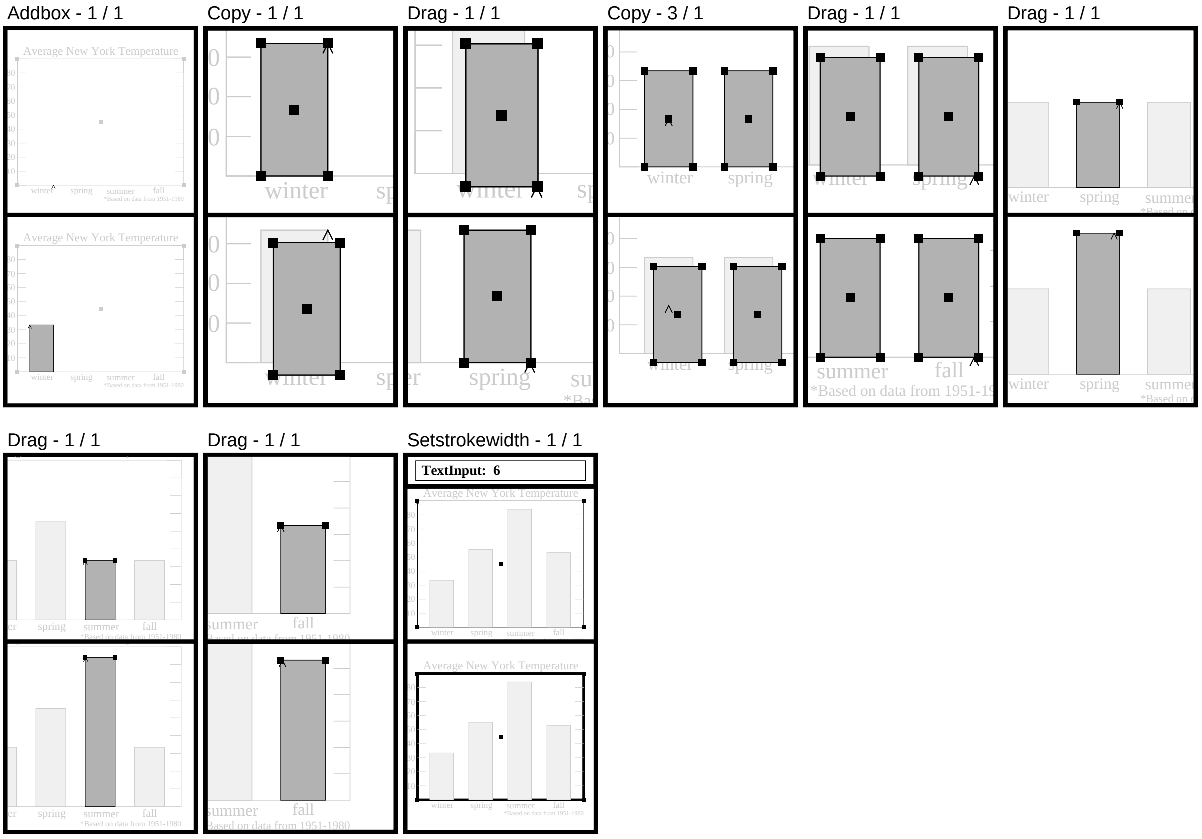



(b)

Figure 3. Collapsing panels. (a) A sequence of history panels prior to a Rotate operation. (b) The same graphical history after the Rotate operation is executed, collapsing the last three panel pairs of Figure 3a.

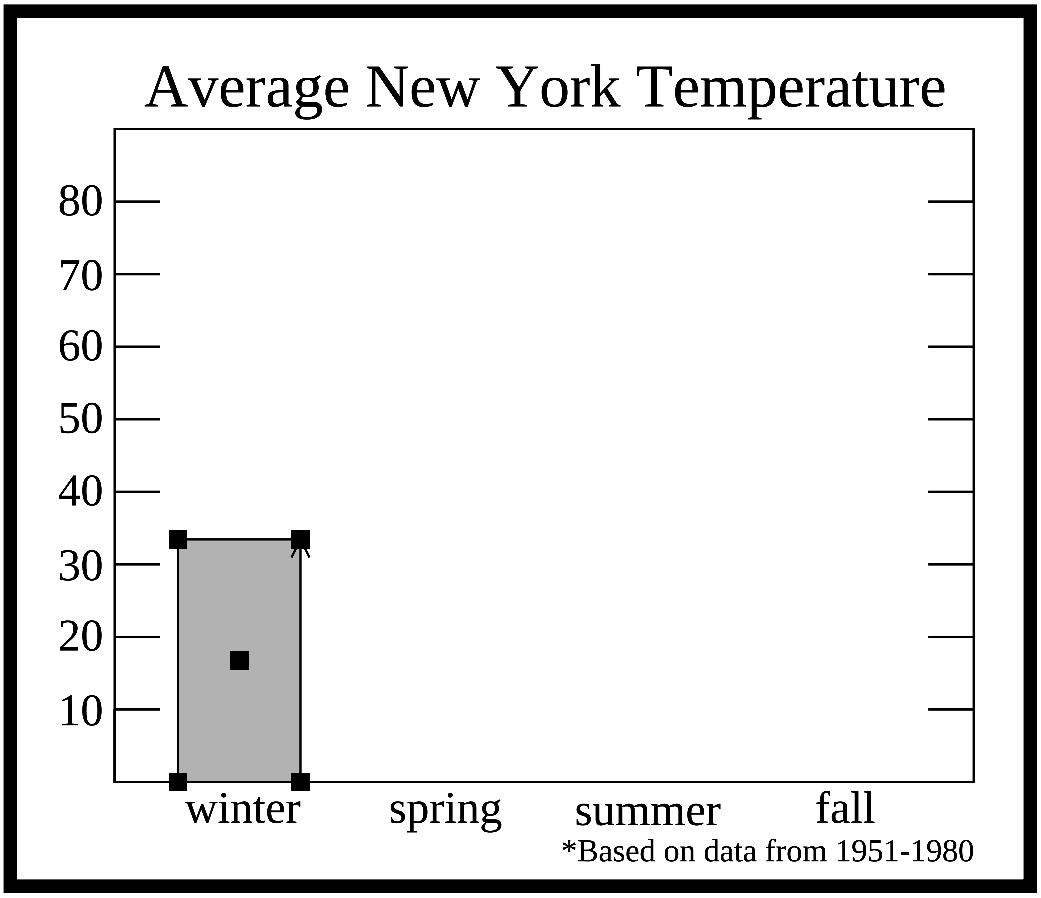



(a)

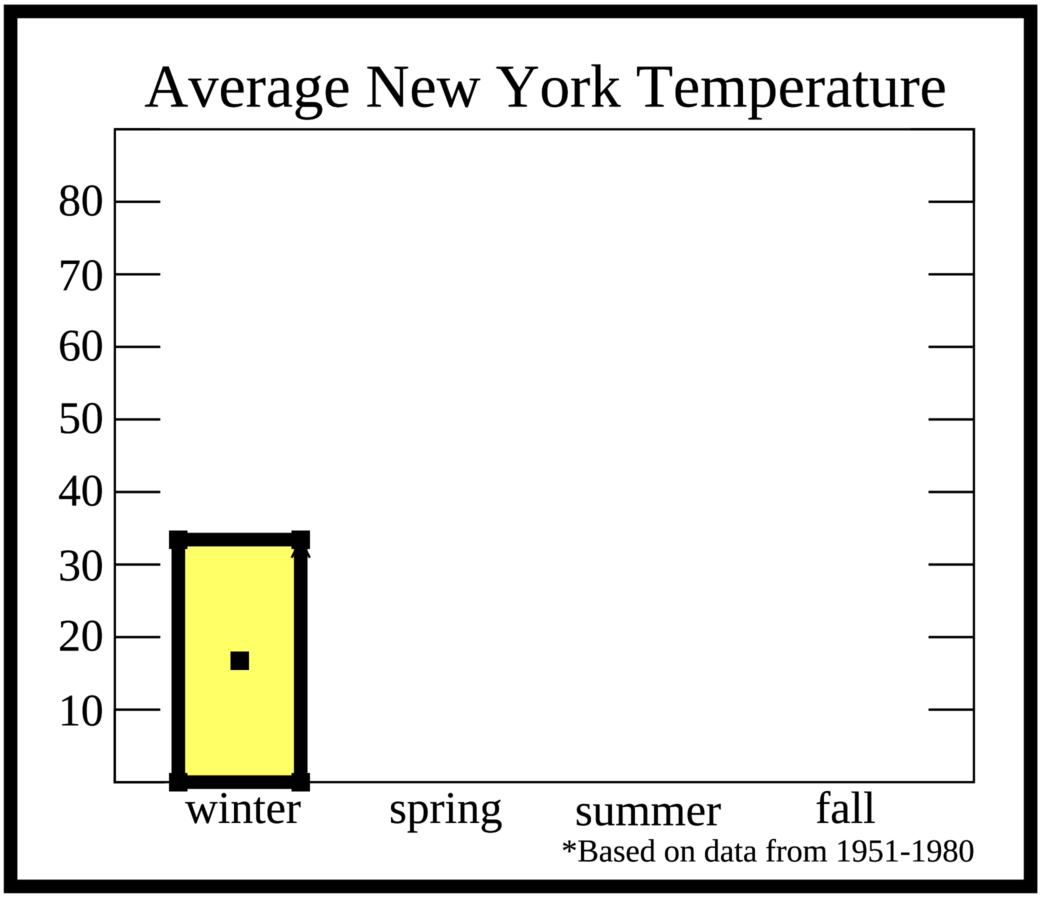



(b)

Figure 4. Bar Chart. (a) The editor window. (b) Its graphical history.

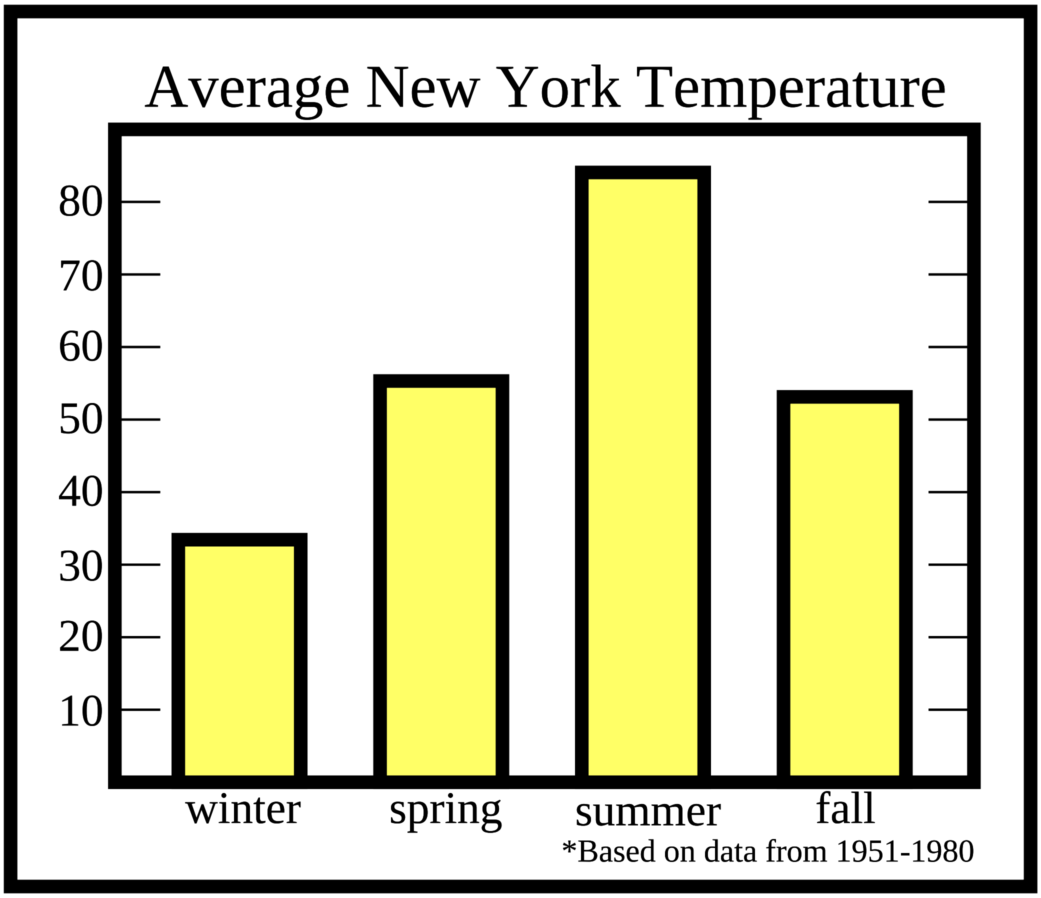



(a)

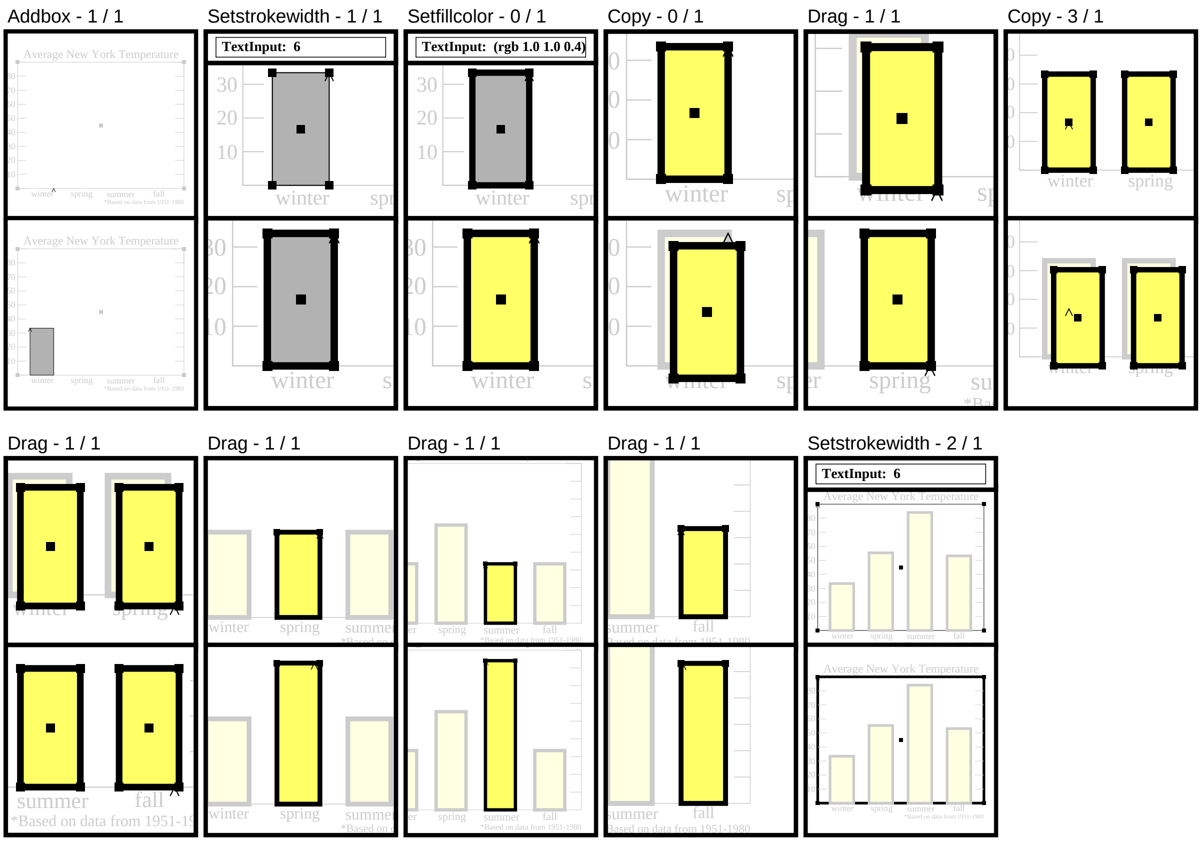



(b)

Figure 5. Undo. (a) The editor window, loaded with the prologue of Figure 4b, panel pair 2. (b) The editor window after modification to the bar's strokewidth and fillcolor.




(a)




(b)

Figure 6. Bar char after redo. (a) The editor window. (b) Its graphical history.