Reducing Repetition in Graphical Editing

David Kurlander

Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, U. S. A.

Abstract

People producing illustrations with graphical editors often need to repeat the same steps over and over again. This paper describes five techniques that reduce the amount of repetition required to create graphical documents, by having the computer play a role in automating repetitive tasks. These techniques: *graphical search and replace*, *constraint-based search and replace*, *constraints from multiple snapshots*, *editable graphical histories*, and *macros by demonstration*, have all been implemented within the Chimera editor framework. Chimera, which contains an object-based editor for producing 2D illustrations, was built as a testbed for this research. All of these techniques are demonstrational or example-based. The user specifies concrete examples of tasks, and the system applies the tasks to other data. In addition to reducing repetition, these techniques allow users to customize the editor for the tasks that they frequently perform, and expert users to encapsulate their knowledge in a form that other users can exploit.

1. INTRODUCTION

People that use graphical editors often find themselves performing the same tasks repeatedly. They may need to change one set of graphical properties to another everywhere in a very large scene. Every occurrence of one shape may need to be edited into another. Often the same set of geometric relationships needs to be established repeatedly over a number of different objects. Every time an object is repositioned, there may be other objects dependent on the first that must be changed as well.

Here I discuss five different techniques that reduce repetition in graphical editing. The techniques are independently useful, but can be combined to form a more effective means of automating repetition. These five techniques have been incorporated into the Chimera editor. Chimera, an application built specifically for this research, is actually an editor framework, in which special purpose editors have been embedded. Currently Chimera includes editor components for creating and modifying 2D object-based graphics, interfaces, and text. The techniques described here work in both the graphics and interface editing components of Chimera. All five techniques are example-based. The user supplies an example of the task to be performed on a sample set of objects, and this example is generalized to work on other inputs and contexts. The user need not have special programming skills to take advantage of these techniques, and these techniques allow users to express complex scene manipulations using concrete examples that are specified using basic graphical editing skills.

Each of the next five sections describe one of these techniques for reducing repetition and promoting extensibility in graphical editors. Section 7 then explains how the techniques are interrelated, and how they can be used together to make a more powerful system for reducing repetition. Due to space limitations, this paper summarizes these five techniques only briefly. Further detail about the techniques, and many examples of their use appear elsewhere [6], and a videotape is also available that demonstrates these techniques at work in Chimera [5].

2. GRAPHICAL SEARCH AND REPLACE

One of the most common types of repetition in graphical editing involves making repetitive changes to shape or graphical properties, such as line style or fill color, many places in a scene. Two traditional graphical editing techniques, instancing and grouping, have proven useful in automating such repetition. Sutherland's Sketchpad system introduced the concept of instancing [9]. The user defines a master object and then instantiates it throughout a scene. Modifications to the master automatically propagate to all of its instances. Many commercial graphical editors, such as MacDraw [2] and Adobe Illustrator [1] allow multiple related objects to be grouped together. Entire groups of objects can be selected and modified as easily as if they were a single object. A disadvantage of both of these techniques is that they require special structuring of the scene to facilitate the process of making repetitive changes. If the user cannot predict, while creating the document, which repetitive changes will likely be required later, or if the user is just too lazy to set up the proper structuring, then these two techniques will not help. Another technique, *graphical search and replace*, can be used in these cases.

Graphical search and replace, is the analogue to textual search and replace in text editors. The user provides an example of a valid match and replacement by copying objects from the scene or drawing new objects. Chimera's graphical search and replace utility is called MatchTool 2. It contains two editor panes: one for the sample search object and one for the sample replacement. The user further refines the search and replace specification by indicating which properties of the search and replace objects are significant. This is done by checking off the significant properties in two columns of checkboxes. A collection of parameters provide further control over the search and replace process. For example, when rotation and scale invariance are chosen, shapes in the search pattern will match similar shapes in the scene at any rotation or scale. The shape tolerance parameter adjusts how close (according to MatchTool 2's shape metric) shapes in the scene must be to those in the search pattern for a match to occur. When the polarity parameter is turned on, two shapes will only match if they were drawn in the same direction. The granularity parameter adjusts how much scene structure is ignored in finding matches, and the context sensitivity parameter allows objects matching only a specified subset of the search pattern to be replaced.

This technique can be used for commonplace editing tasks, such as finding all combinations of a particular fill color and line style, and changing the fill color, or finding all occurrences of a particular shape, and changing it. However, it can also be used for a number of other applications. Graphical search and replace can serve as a tool for generating complex shapes formed by graphical grammars. It can also add complexity to simple scenes by replacing components of graphical templates. Graphical search can find graphical scene files by

content rather than by name, when serving as the basis for a graphical grep capability. Also, graphical search can be used as an iteration mechanism for graphical macros. A more detailed discussion of graphical search and replace, as well as many examples of its use, appear in [3] and [6].

3. CONSTRAINT-BASED SEARCH AND REPLACE

Though graphical search and replace facilitates making repetitive changes to shape, it operates on the complete shape of the pattern. There is no way to specify that only a particular aspect of shape (such as a certain angle or line length) is of interest. A second technique, *constraint-based search and replace*, adds greater selectivity to the search and replace specification. It allows searches and replaces on more specific geometric relationships by having constraints indicate which geometric properties are significant.

Geometric constraints can appear in both the search and replacement examples of constraint-based search and replace specifications. Constraints in the search pattern indicate which geometric relationships must be obeyed by scene objects for them to match the search pattern. For example, we can search for nearly connected lines by providing a search pattern containing two lines connected by a coincident vertex constraint. The tolerance of the search is also provided by example. For example, if the two lines in the search pattern are really a quarter inch apart (breaking the coincident vertex constraint), then all pairs of lines in the scene that are no greater than a quarter inch apart at their endpoints will match the pattern.

The replacement pattern can contain two different classes of geometric constraints. One class of constraint indicates new relationships to be established in each match when a replacement is performed. For example, if the replacement example also contains two lines connected by a coincident vertex constraint, then all nearly connected lines matching the search pattern will be connected together precisely by the replacement. A second class of constraint in the replacement pattern specifies which geometric properties of the match *should not* be changed by the replacement. For example when performing this replacement to connect nearly connected lines, it may be important that locations of the other endpoints of the lines be unchanged, or that the length of the lines not be modified. This second class of constraint specifies which geometric relationships originally in the match cannot be changed in the process of establishing the new relationships.

Constraint-based search and replace can perform very useful scene transformations, such as making all nearly horizontal lines truly horizontal, or adjusting all nearly 90 degree angles to be precisely 90 degrees. This technique can be used for illustration beautification, but whereas previous beautification systems were not extensible, or required programming to add new rules [8] [7], this technique allows the end user to extend the beautification rule set without programming, using an example-based technique. Also, using constraint-based search and replace, end users can define rules that not only beautify existing scene objects, but also add objects to the scene, constrained to existing objects in interesting ways. For example, constraint-based rules can be written to wrap rectangles around text strings, or make right angles rounded by splicing in arcs so that tangent continuity is maintained at the arcs' endpoints. Constraint-based search rules can be archived together in rule sets. Sets of rules in a ruleset can be activated and applied together to a scene. Constraint-based search and replace simplifies the process of making repetitive changes to object geometry throughout a scene.

4. CONSTRAINTS FROM MULTIPLE SNAPSHOTS

With constraint-based search and replace, a graphical editor can find intended geometric relationships in a static scene and enforce these relationships. However in graphical editing, many important object relationships govern how objects are allowed to move in relation to one another, and these often cannot be extracted from a static scene. A third technique, *constraints from multiple snapshots*, is helpful in these cases. The user provides the system with a set of valid configurations (or "snapshots") of scene objects, and the system automatically instantiates constraints that are present in each configuration. Constraints reduce the amount of repetition that users perform in graphical editing, by automatically re-establishing important geometric relationships when some scene objects move. Constraints from multiple snapshots provides a means of specifying constraints that is easier to use in some cases than traditional declarative specification.

The user need not provide, in advance, a complete set of snapshots that unambiguously determines the intended constraint set. The process of adding new snapshots, like traditional constraint specification, can be accomplished incrementally. For example, the user might initially provide two example snapshots that easily come to mind. When the user then turns on constraints and tries to manipulate the scene objects, the constraints that are present in both snapshots will be instantiated, and will restrict the objects' motion. If the user then notices that these constraints preclude another desired configuration, the user can turn off constraints, provide this configuration as an additional snapshot, and all of the constraints interfering with this configuration will be removed automatically.

Chimera has a very efficient algorithm for inferring constraints from multiple snapshots. Constraints in Chimera apply to object vertices, and as objects in Chimera are transformed, the editor monitors transformations that are applied to each vertex. Vertices that have been transformed together since the very first snapshot are placed together in a transformational group. It is very easy to determine which constraints, from our collection of generally useful geometric relationships, should be instantiated on sets of vertices in a transformational group, and which should not. For example, if a set of vertices has only been rotated together since the very first snapshot, then we know that though the slope between each pair of vertices in the group will have changed, the distance between pairs will been invariant. So there is an implied distance constraint between each pair of vertices in this group, but no slope constraints. The complete algorithm and examples of its use are described in [6].

5. EDITABLE GRAPHICAL HISTORIES

A fourth technique, *editable graphical histories*, is a means of visually representing sequences of commands in a graphical user interface. In the character-oriented world, applications typically present their histories as a textual list of commands. This technique does not extend well to the graphical domain, where screen position and other non-textual properties, such as shape, are poorly represented in textual terms. Some graphical applications animate the history directly in the application canvas, but such histories can be difficult to understand and edit, because at any single moment, no temporal context is accessible, and also because the presentation lacks structure. Editable graphical histories use a comic strip metaphor to depict the important operations in a session with a graphical application, in this case the Chi-

mera editor. As application commands are invoked, new panels appear in the history window depicting the commands. The panels use the same visual language as the interface itself, so they are easily understood by people familiar with the interface.

To make the histories more readily understood, we employ three techniques. Related operations are coalesced into individual panels representing logical operations rather than physical operations. This also makes the history more compact. Panels show only those parts of the scene relevant to the operations they represent. Also objects in the panels are rendered according to their role in the explanation. These histories form an interface to an undo mechanism, whereby the user can select any panel and have the system restore the editor state back to that point in time. The histories also reduce repetition by allowing the user to select a series of previously executed operations and redo them. The histories can also be made editable, allowing the user to edit the scene as it existed at any time. Editable graphical histories are further described in [4] and [6].

6. GRAPHICAL MACROS BY EXAMPLE

These histories work in conjunction with a fifth technique: *graphical macros by example*. At any point in time, the user can scan through the graphical history and select a generally useful sequence of commands to be turned into a macro. A new Macro Builder window appears, containing copies of the selected panels. The user then declares each argument by making the panels editable, selecting a copy of the argument anywhere it appears in the panels, and executing the Make-Argument command. For each argument declaration, a new panel appears at the beginning of the macro showing the selected argument and its name.

Playing back the recorded commands verbatim is of limited use, since commands often work as intended only in contexts very similar to the one in which they were originally demonstrated. Chimera's macro facility provides a more powerful mechanism for reducing repetition by generalizing each of the macro's commands to work in other contexts. Chimera will choose a default generalization of each command, according to a built-in set of heuristics, but the user can view these generalizations and override them if necessary. The macro system relies on the graphical history representation to provide support for selecting commands, parameterizing the macro, editing and debugging its contents, and generalizing it to work in different contexts.

7. CONCLUSION

The five techniques described here for reducing repetition in graphical editing tasks are all implemented in Chimera, and they support one another synergistically. Graphical and constraint-based search and replace, for example, can work together, since a single replacement specification can modify geometric and graphical properties. Both kinds of search and replace can be used as an iteration mechanism for graphical macros. For example, a macro can be invoked on all text strings, or all right angles. Editable graphical histories also provide the visual representation for Chimera's macro by example facility.

If Chimera's implementation were extended, there are a number of other ways that these techniques could work together. Currently search and replace operations and snapshots do not

appear in the graphical history, but they could and should. The history mechanism includes landmark objects in the panels to help communicate which region of the scene the panels represent. Currently Chimera has an *ad hoc* mechanism for choosing landmarks. Good landmarks are distinct, and graphical search could potentially be used to determine whether prospective landmarks are unique in the scene.

All of these techniques, with the exception of graphical search, trivially can be extended to graphical editing in three dimensions. The algorithms used for graphical search would need to be supplemented to allow searches on shapes formed by surfaces. It would also be interesting to explore how these techniques might be applied to domains other than graphical editing.

Acknowledgments

Steven Feiner provided helpful advice throughout this project, and supervised my research while I was a doctoral candidate at Columbia University. Eric Bier proposed that I investigate graphical search and replace while I was spending a summer at Xerox PARC, and Eric collaborated with me on that component of this research.

References

1. Adobe Systems Inc. *Adobe Illustrator User Manual*. Macintosh version 3. Part no. 0199-2045 rev. 1. 1585 Charleston Road, Mountain View, CA 94039. November 1990.
2. Claris Corporation. *MacDraw II Reference*. 440 Clyde Ave., Mountain View, CA 94043. 1988.
3. Kurlander, David, and Bier, Eric A. Graphical Search and Replace. Proceedings of SIGGRAPH ‘88 (Atlanta, Georgia, August 1-5, 1988). In *Computer Graphics 22*, 4 (August 1988). 113-120.
4. Kurlander, David and Feiner, Steven. A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands. In *Visual Languages and Visual Programming*, Shi-Kuo Chang, ed. Plenum Press, New York. 1990. 257-275.
5. Kurlander, David. Graphical Editing by Example: A Demonstration. Videotape. Columbia University. March 1992. To appear in a 1993 issue of the *SIGGRAPH Video Review.* Abstracted in INTERCHI ‘93 Conference Proceedings.
6. Kurlander, David. Graphical Editing by Example. Ph. D. Dissertation. Department of Computer Science. Columbia University. 1993.
7. Myers, Brad A. *Creating User Interfaces by Demonstration.* Academic Press, Boston, 1988.
8. Pavlidis, Theo and Van Wyk, Christopher J. An Automatic Beautifier for Drawings and Illustrations. Proceedings of SIGGRAPH ‘85 (San Francisco, CA, July 22-26, 1985) In *Computer Graphics 19,* 3 (July 1985). 225-234.
9. Sutherland, Ivan E. SketchPad: A Man-Machine Graphical Communication System. AFIPS Conference Proceedings, Spring Joint Computer Conference. 1963. 329-346.