Graphical Search and Replace

David Kurlander
Computer Science Department
Columbia University
New York, NY 10027

Eric A. Bier
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304

Abstract

Graphical search is a technique for finding all instances of a graphical pattern in a synthetic picture in which objects are regions bounded by lines and curves. The pattern may describe shape, color and other properties. Matched objects may be allowed to differ from the pattern in rotation and scale or may differ in shape by a specified tolerance. *Graphical replace* is a technique for replacing the shape, color, or other properties of matched objects with new properties described in a replacement pattern. Combined, the two techniques are similar to textual search and replace in text editors. Graphical search and replace can be used to make global changes to illustrations with repetitive patterns, independent of the means used to make those patterns. It can also be used to create a class of iterative or recursive shapes that can be specified by replacement rules.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques – interaction techniques; I.5.4 [Pattern Recognition]: Applications – graphical editing

Additional Keywords and Phrases: Search and replace, graphical editing, curve matching, graphical grammars, graphical macros

1. Introduction

Most graphic arts quality illustrations contain some degree of coherence. For example, the same font, color or stroke width is used throughout a set of shapes, or a particular shape is used repeatedly at different translations, rotations, or sizes. Changing one of the coherent properties of an illustration (e.g., changing all red circles into orange ellipses), requires making the change throughout the illustration. Pictures can be structured to make such changes easy. For example, some editors allow objects to be grouped into a cluster that can be selected as a unit, making it easy to change properties of the clustered objects all at once. Other editors allow the user to declare that an object is an instance of a library object; changes to the library object are reflected in all of its instances at once. However, both of these techniques require the user to decide at an early stage what properties will need to be edited coherently, and to structure the illustration accordingly. We propose an alternative technique, *graphical search and replace*, that allows graphical scenes to be edited coherently without special structuring of the illustration.

Graphical search and replace works much like text substitution in a word processor. The user first describes a search pattern composed of a set of synthetic shapes (regions bounded by lines, conics, and splines), a set of style properties such as stroke width, stroke color, and fill color, and a set of search parameters including an error tolerance and an indication of whether or not a match may be a rotation of the pattern. Next, the user describes a replacement pattern made up of synthetic shapes and style properties. The graphics editor then searches in the illustration for an object that matches the pattern. If the user requests a replacement, the editor replaces the shape, changes its style properties, or both. This can be done one shape at a time, in a top to bottom, left to right order, or can be performed on all matching shapes at once.

Graphical search and replace has a number of applications. It can be used to make changes to many objects at once. If the replacement pattern is more elaborate than the search pattern, these multiple changes can turn a simple repetitive picture, used as a template, into an elaborate composition. Furthermore, if the replacement pattern contains parts that match the search pattern, graphical search and replace can be used repeatedly to build up a recursive shape. If an editing macro is performed on each set of matched objects instead of a replacement, graphical search leads to an even more general way to modify illustrations. Finally, if performed on multiple files, graphical search can be used to find those picture files containing specified graphical features.

Graphical search and replace is related to a wide variety of other topics. It can be viewed as an interface to a graphical database, where typical transactions include queries for objects with a particular set of graphical properties and modifications of these properties. The representation of graphical information in databases has been studied [7, 15]. Graphical search and replace can also be viewed as a method for specifying graphical grammars that generate recursive shapes. Shape grammars, a subset of graphical grammars, have been analyzed extensively [5, 12], and grammars have been used in graphics to make realistic imagery [11]. Pattern recognition algorithms have been used to search for occurrences of a particular shape [6, 8]. Some of these algorithms are appropriate for use in graphical search and replace. Instancing was used in the earliest of drawing systems, Sketchpad [13], and continues to be used in many current systems. Graphical search and replace is proposed as an alternative to instancing for producing coherent changes in graphical documents.

Graphical search and replace has been implemented in the MatchTool, a companion to the Gargoyle two-dimensional illustrator [1, 10] running in the Cedar programming environment [14] on the Xerox Dorado high-performance personal workstation [9]. The MatchTool is similar in user interface to the EditTool, a textual search and replace tool that works with Tioga, the Cedar text editor.

to edit multi-font tree-structured documents. All figures in this paper were created with the MatchTool and Gargoyle.

In section 2, we describe the MatchTool from the user's point of view, showing graphical search and replace in action. In section 3, we describe the implementation of the MatchTool, including the control structures needed to search the scene in top to bottom, left to right order and the algorithm that determines whether or not two objects match. In section 4, we discuss some of the applications of graphical search and replace. In section 5, we present our conclusions and future research directions.

2. User Interface

2.1 Search and Replace for Shapes

In this section, we describe how graphical search and replace can be used to make changes to an illustration. We need two windows on our workstation display. In the Gargoyle editing window is the illustration that we are editing. In the MatchTool window, we specify search and replace requests. At the top of the MatchTool are two panes (sub-windows), each of which is a small Gargoyle editing window. In the first of these, the Search Pane, we construct the pattern that we wish to search for in our illustration. In the second of these, the Replace Pane, we construct the replacement pattern.

Figure 1 shows an illustration of a map that is being edited in a Gargoyle window. This is a synthetic picture; the highway sign borders are represented as arcs, line segments and parametric curves, and the roadways are parametric curves. The text is represented as an ASCII string, a font name and an affine transformation. Recently, sections of Highway 17 have been renamed Interstate 880. We can use the MatchTool to make the substitutions.

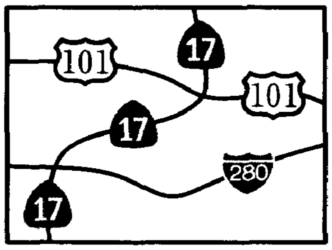


Figure 1. An illustration of the freeways near San Jose, California.

Figure 2 shows the two panes of the MatchTool. In the Search Pane, we put a Highway 17 sign, copying it from the illustration. In the Replace Pane, we put an Interstate 880 sign, drawing it in place or copying it from a different illustration. We make sure that the centers of both signs are at the same coordinates, so that no offset will be introduced when replacements are performed. The centers can be aligned, for instance, by positioning both signs in the Search Pane, and then moving the Interstate 880 sign into the Replace Pane with a "move" operation that preserves coordinates.

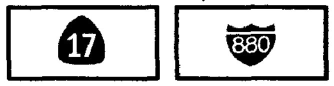


Figure 2. Search and replacement patterns in the MatchTool.

The MatchTool user interface has four buttons that initiate search actions: Search, Yes, No, and ChangeAll. If we press the Search button at this point, the MatchTool will search the illustration in top to bottom, left to right order, looking for Highway 17 signs. When it finds the topmost one, it selects it. If we then press the Yes button, the MatchTool will delete the Highway 17 sign, add an Interstate 880 sign, and initiate another search. This search finds the second sign. Pressing Yes again, we replace the second Highway 17 sign and select the third. Since Highway 17 has only been renamed north of Interstate 280, we are done. If we press the No button at this point, the MatchTool will leave the third Highway 17 sign as it is and report that there are no further matches. The resulting picture is shown in Figure 3. Had we wished to replace all of the Highway 17 signs at once, we could have used the ChangeAll button.

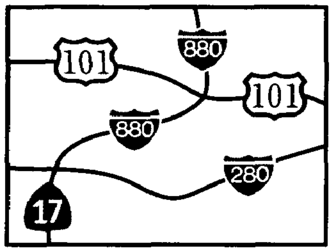


Figure 3. The San Jose freeways after the northern sections of Highway 17 have been relabeled.

In the example above, we searched for a collection of shapes of known size, shape and orientation. Less restrictive searches can be achieved by varying one or more of six search parameters—Granularity, Rotation Invariance, Scale Invariance, Polarity, Context Sensitivity, and Tolerance. The user interface for these features is shown in Figure 4. The first four search parameters will be discussed in this section, the last two in section 2.3.

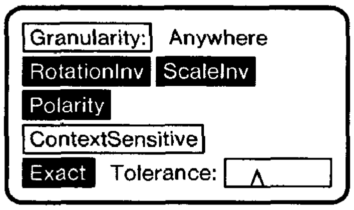


Figure 4. The six search parameters. Options that are white on black are active. All of the regions bordered with a black rectangle are mouse-sensitive.

Granularity may take on the values "cluster", "object", or "anywhere". It tells the MatchTool how much of the structure of the illustration may be ignored when performing matches. If Granularity is set to "cluster", then a group of objects that have been clustered in Gargoyle will only match a similar complete cluster in the MatchTool; the individual objects in the cluster cannot match separately. At the "object" granularity, a Gargoyle object A will match a similar object B in the pattern, even if A is part of a cluster. At this granularity, A must be matched in entirety; a pattern containing only a subset of A's parts will not match. At the "anywhere" granularity, parts of an object A may be matched by a pattern object B if all of B's parts match corresponding parts in A. At this granularity, an entire object in the MatchTool search pane can match a portion of a single object in the editor scene.

In an "anywhere" match, the lowest-level scene elements, segments, are treated as atomic; it is impossible to match on parts of them. This certainly has performance benefits, but also avoids other problems inherent in replacing portions of particular object classes. For example, it is impossible to replace portions of some segment types, such as non-local splines, without potentially causing changes to the entire segment.

When Rotation Invariance is turned on, the pattern matches a configuration of scene objects if some combination of translation and rotation will bring the pattern and configuration into correspondence. If more than one rotation is possible, the MatchTool will choose the smallest rotation that works. When Scale Invariance is turned on, the pattern matches a configuration of scene objects if some combination of translation and scaling will bring the pattern and configuration into correspondence. Likewise, when both Rotation Invariance and Scale Invariance are on, the MatchTool will try to use combinations of translation, rotation, and scaling to bring the pattern into correspondence with the scene objects.

When Polarity is on, two curves will only match if they were drawn in the same direction. For instance, if a rotation-invariant match is performed where the pattern is a straight line segment, the pattern will match any straight line segment that has the same length as the pattern. With Polarity off, each match can be made with two rotations, differing by 180 degrees. With Polarity on, the MatchTool will take into account the direction in which each line segment was drawn, and will choose the rotation that aligns those directions.

We can use these search parameters to construct a triadic Koch snowflake. We draw an equilateral triangle, constructing the line segments in clockwise order, giving us Figure 5. Next, we select one edge of the triangle and copy it into the Search Pane. In the Replace Pane, we draw four line segments each 1/3 the length of the original. These line segments are drawn in the same direction (lower left to upper right) as the original segment. We must also be careful that the replacement shape begins and ends at the same coordinates as the original segment, so that when scene objects are replaced, no offset will be introduced. The resulting Search and Replace Panes are shown in Figure 6.

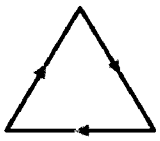


Figure 5. An equilateral triangle. Grey arrows show the directions in which the edges were drawn.

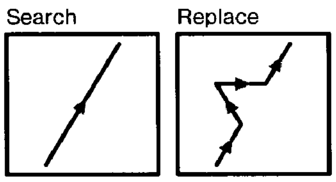


Figure 6. Preparing to replace a line segment with four line segments. Grey arrows show the directions in which the edges were drawn.

We set Granularity at "anywhere" so that the search can find individual segments of the original triangle. We turn on Rotation Invariance. We turn on Polarity. We press ChangeAll. All of the line segments in the original triangle are replaced by four-segment paths as shown in Figure 7(a). If we hit ChangeAll again, nothing happens, because the picture no longer contains any segments that are the same length as the pattern segment. If we turn on Scale Invariance and try ChangeAll again, all of the segments of Figure 7(a) are replaced by four-segment paths to produce Figure 7(b).

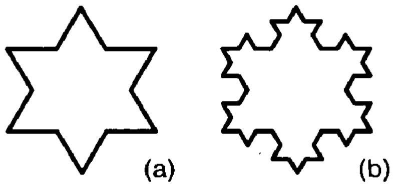
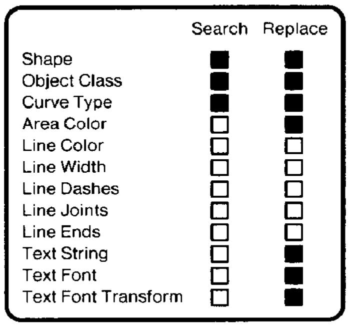


Figure 7. Triadic Koch snowflakes. (a) After one ChangeAll operation using the search and replacement patterns of Figure 6. (b) After a second ChangeAll, with Scale Invariance turned on.

2.2 Search and Replace for Graphical Style

Often, we will want to search for properties of graphical objects other than shape, such as object class – Box, Circle, or Polygon; curve type – line, Bézier, B-spline, natural spline, conic, or arc; area color; line properties – line color, stroke width, dash pattern, joint blending, or stroke end shape; or text properties – ASCII string, font family, or font transformation. Likewise, we may be interested in replacing properties of the matched objects other than shape. Figure 8 shows a portion of the MatchTool control panel called the Search Column and the Replace Column. The black squares in the Search Column indicate those properties of the objects in the Search Pane that must agree with the scene objects for a match to succeed; other properties can be ignored. The black squares in the Replace Column indicate those graphical properties of the objects in the Replace Pane that will be applied to the matched objects when a replacement is performed; other properties are left alone unless they are determined by the specified properties. The user can toggle each square between white and black by clicking on it with the mouse.



	Search	Replace
Shape	■	■
Object Class	■	■
Curve Type	■	■
Area Color	□	■
Line Color	□	□
Line Width	□	□
Line Dashes	□	□
Line Joints	□	□
Line Ends	□	□
Text String	□	■
Text Font	□	■
Text Font Transform	□	■

Figure 8. The Search Column and Replace Column.

When the search and replacement properties are specified appropriately, the MatchTool can be used to change the shape of an object while leaving its color unchanged. For example, to make a multi-colored snowflake, we might start with a triangle as before but give its three edges different colors, as shown in Figure 9(a). If we turn off Line Color in the Search Column, then all three edges will still match the pattern we used above. If we turn off Line Color in the Replace Column, then as each edge is replaced, its color is applied to the replacement shape. If we use the same Search Pane and Replace Pane as before, then after two ChangeAlls, we get the picture in Figure 9(b). The MatchTool can also change the color of an object while leaving its shape unchanged.

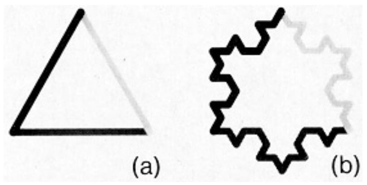


Figure 9. Replacing the shape of the edges of a triangle while leaving the colors as they were.

2.3 Advanced Search and Replace

In this section, we discuss the last two search parameters – context-sensitive search, and variable error tolerance.

Context-sensitive search allows the user to search for the occurrence of a set of shapes, A, in the presence of another set of shapes, B. Only the shapes that match A are selected and eligible to be replaced. To perform a context-sensitive search, the user places all of the pattern shapes, A and B, in the Search Pane and indicates which shapes are in set A by selecting them. The user then turns Context Sensitivity on and initiates a search.

Context-sensitive search, by reducing the set of shapes that are eligible to be replaced, can remove ambiguities present in certain pattern specifications. For instance, if the Shapes property in the Replace Column is on but the Line Color property is off, the MatchTool must replace the matched shapes with the replacement shapes, copying the line color of the matched shapes to the replacement shapes. A problem occurs if the search pattern matches a set of shapes that have several different line colors; we don't know which objects in the replacement pattern should receive which line colors from the match. To solve this problem, we can break up the search into several context-sensitive searches, each of which will only select objects of a single line color.

Figures 10 and 11 give an example of this use of context-sensitive search. The user starts with a picture of a gumball machine containing softballs and replaces the softballs by footballs. Each football will take its orientation from the stitching direction of the softball it replaces and will take its area color and the line color of its stitching from this softball as well. Because the line color of the stitching and the line color of the outer circle are different in a single softball, we cannot perform this replacement all at once. Instead, we perform one context-sensitive search and one regular search, both shown in Figure 10, to get the desired result, shown in Figure 11. The first search replaces circles (in the presence of stitching) by footballs with the proper orientation, area color and line color. The second search replaces the stitching by football stitching with the proper line color.

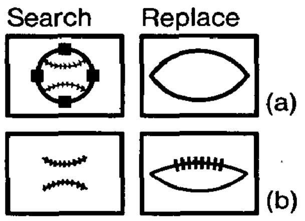


Figure 10. (a) Search for a circle in the context of stitches and replace with a football outline. The black boxes on the circle indicate that it is selected. (b) Search for softball stitches and replace by football stitches.

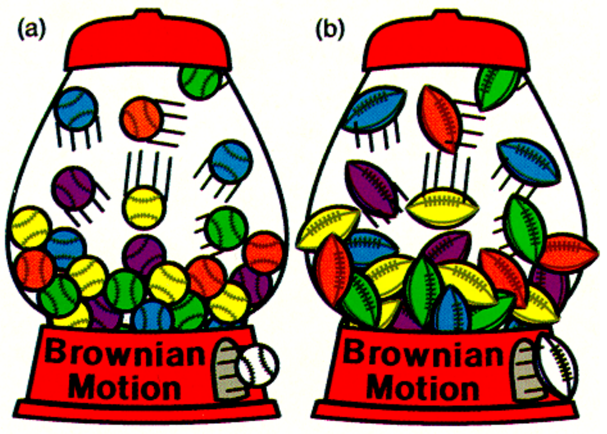


Figure 11. (a) A gumball machine filled with softballs, used as a T-shirt design for a softball team. (created by Polle Zellweger and Jock Mackinlay) (b) The softballs are replaced by footballs.

Variable error tolerance allows the user to find shapes that match the search pattern approximately, but not exactly. By adjusting a slider, the user can increase or decrease the amount of error that the matching algorithms will permit when identifying matches. For example, an appropriate tolerance and search pattern would enable matches of all shapes that approximate circles or all shapes that approximate straight lines. To successfully use this feature, the user must understand some of the details of the shape matching mechanism and must determine an appropriate tolerance by trial and error. One use of inexact matches is described in section 4.4 on graphical grep.

3. Implementation

3.1 Control Structures

Since the data in text documents is ordered in a linear fashion, and graphical data tends to be distributed over at least two dimensions, the control structures for a graphical search and replace program must be more elaborate than those of its textual analog. In textual search and replace, very little state information needs to be kept. A current location specifies the point from which the search will proceed, and when a replacement is made, there is no ambiguity over where the replacement text will be placed. In addition, there is a clear choice for the order in which items will be searched provided by the linear order of the character stream. The mechanism for graphical search and replace is more complex.

We search graphical objects in a top to bottom, left to right order. This search order is familiar to users of textual search and reduces screen updates because matches located close to one another in the scene tend to be found together. Reverse searches proceed in exactly the opposite order from forward searches. The Gargoyle software cursor, the *caret*, represents the current location of the search. In a forward search, only objects below the caret are considered, and in a backward search, only objects above the caret are considered. The user can move the caret in the Gargoyle scene in order to direct the search.

Before searching begins, a snapshot is taken of all scene objects that may participate in the search. This prevents the search from finding objects that are added to the scene by a replace operation in progress. As a result, replacement operations are predictable and will always terminate. The snapshot is a singly-linked list, called the *search list*. Each list element is either a cluster of scene objects, if Granularity is set to "cluster", or an individual object, if Granularity is set to "object" or "anywhere". The list is ordered by the upper left corner of the bounding box (a tight-fitting rectangle aligned with the coordinate axes) of each element. For a forward search, the elements are in top to bottom, left to right order. For a backward search, the order is reversed. Updated snapshots of the scene are taken whenever an entirely new search is initiated. A search is considered "entirely new" if, since the last search operation, the caret has been manually repositioned, the scene has been edited, a new Gargoyle window is being searched, the pattern has been modified, or the search direction has changed.

The current search pattern is represented by a list of lists, called the *pattern list*. Each element of the pattern list is a list of property-value pairs representing the relevant properties of one of the graphical objects in the Search Pane. The possible properties appear in Figure 8. All of the property-value lists will include the same set of properties, namely the set that is active in the Search Column. The pattern list is recomputed at the beginning of a search if the Search Pane or Search Column have been modified since the last search.

Searching involves trying to find a correspondence between elements of the pattern list and elements of the search list. To begin, we choose one element of the pattern list, called the *leading object*, and compare it against the members of the search list in order, starting with the member nearest the current search position. The algorithm that we use to compare two curves for shape equality will be discussed in section 3.2. Once we have found a match for the leading object, we attempt to find matches for all of the other elements in the pattern list. If we succeed, we are done. Otherwise, we try to match the leading object differently. If Rotation Invariance is on, we try matching the leading object against the same object in the search list at a different orientation. When all possible orientations are exhausted, we move on to the next object on the search list.

If we are matching on shape, we use two techniques to improve performance. First, when we have found a match for the leading object, we know where in the scene to look for the remaining shapes in the pattern list. We use bounding boxes to quickly rule out many of the objects on the search list. For instance, we compute where one point of a given pattern object would have to match in the scene and rule out all objects on the search list whose bounding boxes do not contain that point. If the search is not exact, we enlarge the bounding boxes by an amount proportional to the tolerance before testing the point for inclusion. Furthermore, the search list is ordered by the upper left hand corner of the bounding boxes, so we can quickly rule out entire sections of the search list. Second, we choose a good leading object. If possible, we choose an open curve to be the leading object because open curves can match other curves at no more than two different orientations. If all of the pattern objects are closed curves, we choose the curve with the least number of potential matching orientations (see section 3.2).

When Granularity is set to "anywhere", the search mechanism is more elaborate. Objects in the pattern list are matched against both entire objects and portions of objects in the search list. When a match is found, information is saved indicating precisely where the search terminated, so the next search can continue with another part of the same search list object if any unexamined parts remain. Note that it is possible to invoke a search on an object, part of which has already been changed by a prior search and replace.

Let there be m objects in the editor scene and n objects in the search pattern. The worst case complexity of the search algorithm to find all matches of the pattern in the scene is $O(m^2n)$ object to object comparisons, assuming that the search is being made at either the "cluster" or "object" granularity, and the leading object matches no scene object at more than a constant number of orientations. If the granularity of the search is set to "anywhere", then we are effectively matching against a greater number of scene objects, since each object and its eligible subsets (continuous runs of segments) must be considered in the match process. In this case, we redefine m to be the sum over all objects in the scene of the number of eligible subsets in each object, and the complexity expression remains valid.

The expected number of object to object comparisons required to find all matches of a pattern in the scene is no worse than $O(m^2)$. It is rare for the first few elements of the pattern list to match objects in the scene without a complete match occurring. When a complete match does occur, the scene objects participating in the match are removed from further consideration, so no more than m/n matches can be found. Together, these observations lead to the tighter bound. In addition, the use of bounding boxes to narrow down the matching process for shape searches does much to speed up the search. In practice, we have found the speed of this algorithm to be acceptable for our applications.

After a match is found, the search list is updated to disallow future matches on the same objects. The objects in the Gargoyle window that were found by the search are selected, and all other objects are deselected. Selection performs a dual function. First, the selection feedback indicates to the user which set of objects has been matched. Second, it prepares the matched objects to be modified by any of the Gargoyle operations that act on selected objects, including deletion, color changes, and transformations. The caret is relocated to the position of the match. If a Yes or ChangeAll is in progress, a replacement or macro operation will be performed at this point. Macro operations are described in section 4.3.

If a replacement is to be performed, we examine the Replace Column. If we are replacing only non-shape properties, the values of these properties are extracted from the shapes in the Replace Pane and applied to the matched objects. If we are replacing shape, the matched objects are deleted and the objects in the Replace Pane are copied into the scene. The new scene objects inherit from the matched shapes the properties not specified in the Replace Column. The new scene objects are positioned in the scene as follows: If we are matching on shape, we have found a transformation that maps the pattern objects onto the matching scene objects; we apply this same transformation to the Replace Pane shapes. Otherwise, we position the replacement so that the center of its bounding box coincides with the bounding box center of the match.

3.2 Curve Matching

At the core of our searching algorithm is a set of routines for comparing two curves for equality. We wish to be able discover that two curves are the same even if the two curves are at different sizes and orientations and even if the curves have different representations. For example, one curve might be a B-spline and the other a collection of Bézier cubic pieces, or one curve might be made of two small arcs and the other a single large arc. Our method is simple and general at the expense of performance. We discuss a technique for improving performance in section 5.2.

To compare two curves, we begin by approximating each curve by a piecewise linear path—a *polyline.* We construct polyline approximations adaptively, so that areas of high curvature are represented by more line segments than flatter areas. To keep polylines from having too many segments, we enforce a minimum length on the polyline segments. Many graphics systems already perform this vectorization, a common step in rendering curves. As shown in Figure 12, the polylines for copies of a curve at different scales may not be scales of one another. The test for equality must tolerate this error (see below).

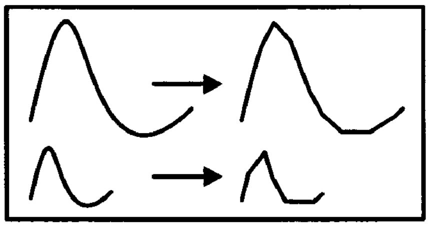


Figure 12. A curve and a scaled down copy of the curve, approximated as polylines. The roughness of the polylines is exaggerated for clarity.

Polylines are transformed to a canonical form so that they can be quickly compared. The nature of the canonical form depends upon whether the match is to be rotation-invariant, scale-invariant, neither, or both, as shown in Figure 13. One point of the polyline is chosen as the *starting point.* For open curves, the first endpoint is used. For closed curves, we use the point of greatest distance from the center of mass of a wire of uniform density lying along the curve (Figure 13(a)). A closed curve may have several points farthest from the center of mass; in this case the curve will have several canonical positions. The polyline is transformed so that its starting point lies at the origin (Figure 13(b)). If a rotation-invariant match is chosen, the polyline is rotated so that the center of mass lies along the positive x axis (Figure 13(c)). If a scale-invariant match is desired, then the polyline of the curve is normalized to have a particular arc length (Figure 13(d)).

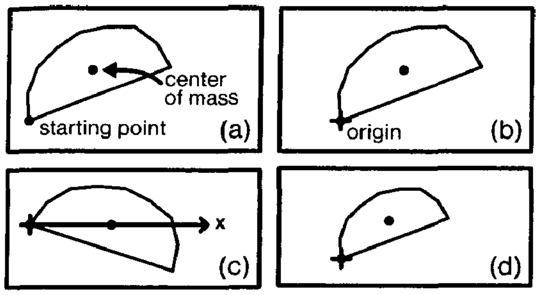


Figure 13. Polyline canonical forms. (a) The original polyline. (b) For all matches, the starting point is translated to the origin. (c) For rotation-invariant matches, the center of mass is rotated onto the positive *x* axis. (d) For scale-invariant matches, the curve is scaled to have a known total arc length.

A set of quick-reject tests can now be applied to the polylines to avoid further computation on pairs of curves that obviously do not match. Several quantities, including arc length, the maximum distance from a curve to its center of mass, and the position of the center of mass relative to the starting point, can now be compared. If these values for two polylines differ by more than a minimal quantity (accounting for floating point error or differences in quantization), then we conclude, without further computation, that the curves do not match.

If these quantities are similar enough, then it is still possible for the two polylines to represent equivalent shapes and a more comprehensive comparison is made. For each vertex of both polylines, we examine the Manhattan (1-norm) distance to the point of parametrically equivalent distance along the other polyline, found by interpolating between vertices if necessary. If this distance ever exceeds a certain threshold the match fails. This threshold, and the quantities used in the quick-reject tests can be adjusted to reflect a user-specified match tolerance.

We use a 1-norm metric because it can be computed quickly, and always produces correct responses with respect to exact shape matches. Other metrics may provide better measures of inexact shape matches, as evaluated by the human eye; however, inexact shape matches are selected relatively infrequently. It may be desirable to have several shape metrics: one for exact matches, and one or more for inexact matches. In the conclusion we mention an inexact shape metric that we are currently investigating. It is important to note that the 1-norm metric together with the quick-reject tests form our shape-matching criteria. In the case of inexact matches, it is possible for curves that would have passed the 1-norm test for equality to fail at least one of the initial tests, and thus be considered unequal.

When comparing two closed curves with the 1-norm metric, we must examine every canonical orientation of one of the curves with a single canonical orientation of the other before declaring a mismatch, unless the arc length or maximum distance to center of mass tests indicate this is not necessary. Although this algorithm for comparing curves is linear with respect to the number, n, of samples in the polylines for all open and some closed curves, there are certain closed shapes that will slow it down to O(n^2). Other representations such as sampling the distance of a curve from its centroid [4] or sampling the curve's curvature [16] can be used to improve this bound.

4. Applications

In addition to coherent changes in illustrations with repeated components, graphical search and replace can be used to make recursive and iterative shapes, to create pictures that have a standard form by modifying graphical templates, to apply graphical editing macros, and to search for picture files based on graphical content. We discuss these applications in this section.

4.1 Graphical Grammars

Graphical search and replace can interactively generate complex shapes described by *graphical grammars*, an extension of shape grammars that allows graphical properties other than shape, such as color and line width, to participate in production rules. The Search Pane and Column specify the left side of a production rule, and the Replace Pane and Column specify the right side. Each replacement operation amounts to a single expansion of the production.

For example, Figure 14 shows the Search and Replace panes for a replacement rule that builds a spiral. We activate area color (in addition to shape and object class) in the Search Column and in the Replace Column. We turn on Rotation Invariance and Scale Invariance. If the initial scene is the same as the picture in the Search Pane, then by clicking the ChangeAll button 27 times, we produce the picture in Figure 15. The innermost copy of the word "MATCHTOOL" is grey.

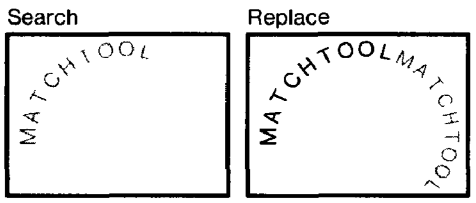


Figure 14. The Search and Replace patterns for a graphical grammar that makes a word spiral.

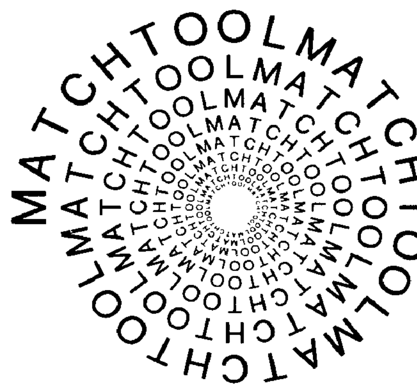


Figure 15. A spiral made of the word "MATCHTOOL" repeated 28 times.

As another example, Figure 16(a) describes a rule that replaces a grey line segment by a brown line segment of the same size, with two grey branches attached. Line Color and Shape are selected in both the Search Column and in the Replace Column. Rotation Invariance and Scale Invariance are turned on. Beginning with a single vertical grey line, we apply the rule five times to produce a leafless brown tree with grey outermost branches. Figure 16(b) describes a second rule that replaces all grey branches by brown branches attached to green leaves. Applying this rule produces the simple graftal tree shown in Figure 16(c). For grammars with more than one rule, it is useful to have multiple pairs of Search and Replace Panes.

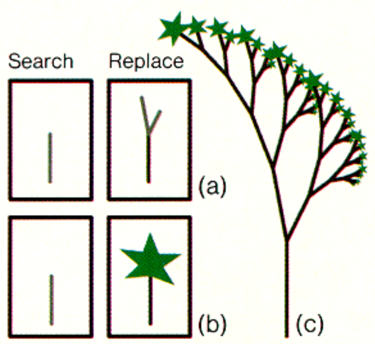


Figure 16. Drawing a binary tree. (a) Replace a grey line with a brown line and two grey lines. Apply this rule five times. (b) Replace a grey line with a brown line and a leaf. (c) The resulting tree.

Figure 17 shows a third example. A triangle is replaced with three triangles, each having half the linear dimensions of the original (Figure 17(a)). If we begin with a single triangle and apply the replacement four times, we get Figure 17(b).

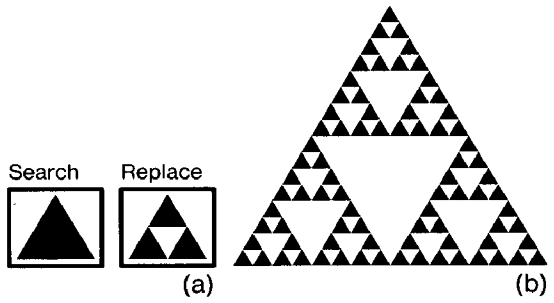


Figure 17. Nested triangles. (a) Replace one large triangle with three smaller triangles. Apply this rule four times. (b) The result.

4.2 Graphical Templates

Graphical search and replace can be used to produce scenes containing complex shapes from scenes containing simple shapes. For example, Figure 18(a) shows three test-tubes containing line segments of different sizes and orientations. The MatchTool is used to replace each segment with an amoeba, using the rule in Figure 18(b). After the large amoeba is colored orange, Figure 19 results. Because drawing line segments is easier than scaling and rotating shapes into place, this picture can be made very rapidly. In this example, Figure 18(a) serves as a graphical template. Many other illustrations could be made from this template by varying the shape in the Replace Pane. Regular polygons make good templates for patterns with cyclic symmetry. Other templates can be made for frieze symmetries and crystallographic symmetries.

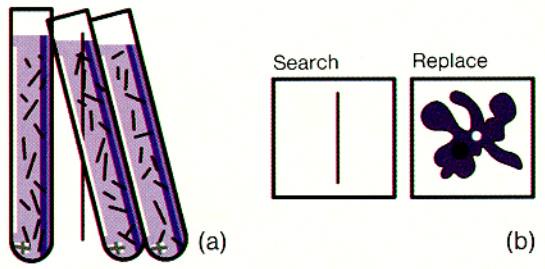


Figure 18. (a) A set of line segments is drawn on a picture of test-tubes. (b) Each line segment is replaced by an amoeba.

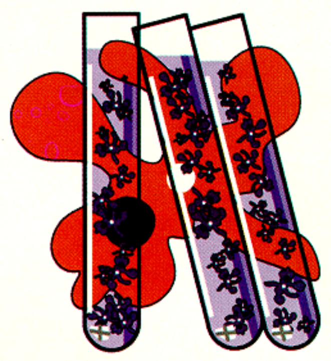


Figure 19. "The Day of the Amoebas."

4.3 Mouse-click Macros

As mentioned above, when a graphical search succeeds in finding a match, the matched objects are selected in preparation for either replacing them or for manually editing them using Gargoyle. The MatchTool provides an alternative to manual editing called *mouse-click macros.* Before invoking a search operation, the user employs the MatchTool to record a macro by performing a set of Gargoyle operations in the Replace Pane. All of the mouse coordinates and button presses are recorded. The user then tells the MatchTool to perform the macro instead of doing replacements and invokes one of the replacement operations (e.g., ChangeAll).

After each match is found, the MatchTool plays back the recorded actions, transforming the mouse coordinates by the same transformation that would have been applied to replacement objects if a replacement were being performed. To prevent unexpected results, the macro commands are restricted to operate only on matched objects.

Mouse-click macros can be used to perform coherent changes that would be difficult with simple replacement. For instance, we can give a set of objects drop shadows by recording a macro that copies an object, colors the copy grey, offsets the copy from the original, and moves the copy underneath.

4.4 Graphical Grep

Grep is a utility in the UNIX^{TM} environment for locating files that contain a particular text pattern. An analogous function for graphical scene files can be implemented on top of graphical search. The user specifies a pattern in the Search Pane and Column and a list of one or more file names, optionally containing wildcards. Graphical grep searches the named files for the specified graphical pattern. The user can choose to list the names of files that contain the graphical pattern or to invoke the editor on the first match, edit the file, and resume the search. The ability to specify a shape tolerance is useful in conjunction with graphical grep. If an exact copy of the graphical pattern cannot be copied from a document, the user can draw an approximate pattern shape by hand.

5. Conclusion

5.1 Summary

We have described a technique for making coherent changes to a graphical scene. The technique requires that the user specify, via a graphical pattern, those objects that are to change, and, via a second graphical pattern, what change is to take place. The two patterns are very similar. Both consist of a pane containing graphical shapes and of a column of buttons describing the properties of interest in the pane. In addition, there are six parameters that are used only for the search operation. This user interface is relatively easy to understand and provides a great deal of power. Furthermore, graphical search and replace can be added to an existing graphical editor with little modification to the editor or its data structures.

Graphical search and replace can accomplish any coherent change that can be accomplished using instancing, and it can be applied more widely. Because it requires no special structuring of the illustration, our technique can be used in editors that do not support instancing and in situations where only a "flat" description of a picture is available. Furthermore, it can be used to modify a collection of objects with identical shapes but different style properties, while instancing schemes tend to require that the instances of a library object be identical in all respects expect for an affine transformation. Graphical search and replace can consider replacements on a case by case basis, allowing changes to some objects but not others.

In addition to making coherent changes, graphical search and replace can be used to make recursive shapes and to copy shapes to positions specified in a graphical template. It extends shape grammars by allowing style properties to appear on each side of the production as well as shapes.

Finally, graphical search can be combined with operations other than replacement. More general coherent modifications can be achieved by playing back a macro on each match. Searching in multiple files provides graphical grep, a means for retrieving graphical documents by content instead of name.

5.2 Future Work

Graphical search could be improved by allowing for more general patterns. We would like to be able to search for all angles of a certain value, or junctions with a certain number of lines radiating from them. We would also like to be able to capture positional relationships such as finding all circles above squares. Such relational metrics have appeared in the literature [3]. We would also like a better way to search for objects that "look like" the search pattern. We may be able to apply string matching techniques [2] to this problem.

Exact pattern matches on curves of known type are an important special case for graphical search. By comparing the curve control points directly instead of comparing polyline representations, we could significantly speed up the majority of searches.

It appears that graphical search and replace can be used as a user interface paradigm for systems that use instancing internally. In fact, for scenes that have an instancing hierarchy it should be possible to speed up graphical search by using pointer comparisons, where possible, in place of geometric comparisons. Likewise, graphical replace can take advantage of instancing by replacing matched objects with library object instances, where possible, instead of allocating new data structures.

Finally, graphical search could serve as the basis for a tool that compares two graphical scene files and reports their differences. Such a tool would be useful for regression testing of graphical editors and for understanding how one picture was changed to make another.

Acknowledgments

We are grateful to Xerox PARC for providing the environment that made this research possible. We would like to thank Ken Pier, Maureen Stone, Subhana Menis, and Jock Mackinlay for comments that lead to an improved paper. We give special thanks to Maureen Stone and Ken Pier for their encouragement during the project.

References

1. Bier, Eric A., and Stone, Maureen C. Snap-Dragging. Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18-22, 1986). In *Computer Graphics 20*, 4 (August 1986), 233-240.
2. Burr, D. J. A Technique for Comparing Curves. In *IEEE Conference on Pattern Recognition and Image Processing* (Chicago, Illinois, August 6-8, 1979), 271-277.
3. Chang, Shi-Kuo, Shi, Qing-Yun, and Yan, Cheng-Wen. Iconic Indexing by 2D Strings. In *IEEE Computer Society Workshop on Visual Languages* (Dallas, Texas, June 25-27, 1986), 12-21.
4. Freeman, Herbert. Shape Description Via The Use of Critical Points. *Pattern Recognition 10*, 3 (1978), 159-166.
5. Gips, James. *Shape Grammars and Their Uses: Artificial Perception, Shape Generation, and Computer Aesthetics.* Birkhauser, Verlag, Basel, Switzerland, 1975.
6. Levine, Martin D. *Vision in Man and Machine*, chapter 10. McGraw Hill, New York, New York, 1983.
7. Palermo, Frank and Weller, Dan. Some Database Requirements for Pictorial Applications. Data Base Techniques for Pictorial Applications (Florence, Italy, June 1979). Edited by A. Blaser. In *Lecture Notes in Computer Science, 81.* Springer-Verlag, Berlin, West Germany, 1980.
8. Pavlidis, Theo. A Review of Algorithms for Shape Analysis. *Computer Graphics and Image Processing 7*, 2 (April 1978), 243-258.
9. Pier, Kenneth A. A Retrospective on the Dorado, a High-Performance Personal Computer. In *Proceedings of the 10th Symposium on Computer Architecture.* SIGARCH/IEEE, (Stockholm, Sweden, June 1983), 252-269.
10. Pier, Kenneth A., Bier, Eric A., and Stone, Maureen C. An Introduction to Gargoyle: An Interactive Illustration Tool. In van Vliet, J.C. (editor), *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography* (EP88), (Nice, France, April 1988), Cambridge University Press, 223-238.
11. Smith, Alvy Ray. Plants, Fractals, and Formal Languages. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23-27, 1984). In *Computer Graphics 18*, 3 (July 1984), 1-10.
12. Stiny, George. *Pictorial and Formal Aspects of Shape and Shape Grammars.* Birkhauser, Verlag, Basel, Switzerland, 1975.
13. Sutherland, Ivan E. Sketchpad: A Man-Machine Graphical Communication System. In *AFIPS Conference Proceedings, Spring Joint Computer Conference, 23.* Spartan Books, Washington, 1963, 329-346.
14. Swinehart, Daniel, Zellweger, Polle, Beach, Richard, and Hagmann, Robert. A Structural View of the Cedar Programming Environment. *ACM Transactions on Programming Languages and Systems 8*, 4 (October 1986), 419-490.
15. Weller, Dan, and Williams, Robin. Graphic and Relational Data Base Support for Problem Solving. Proceedings of SIGGRAPH '76 (Philadelphia, Pennsylvania, July 14-16, 1976). In *Computer Graphics 10*, 2 (Summer 1976), 183-189.
16. Wolfson, Haim. On Curve Matching. Technical Report #256, Courant Institute of Mathematical Sciences, New York, New York, November 1986.