1992

Geometric Search and Replace in Solid Model Editing

Chandrajit L. Bajaj

Kokichi Sugihara

Report Number:
92-078

http://docs.lib.purdue.edu/cstech/998

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
GEOMETRIC SEARCH AND REPLACE IN SOLID MODEL EDITING

Chandrajit Bajaj
Kokichi Sugihara

CSD-TR-92-078
October 14, 1992
Geometric Search and Replace in Solid Model Editing

Chandrajit Bajaj*
Department of Computer Science,
Purdue University,
West Lafayette, IN 47907

Kokichi Sugihara†
Department of Mathematical Engineering and Information Physics,
University of Tokyo,
Tokyo, Japan 113

Abstract

We present an efficient solution to a three dimensional geometric search and replace operation in the editing of solid models in both boundary representation or feature based solid modeling systems. This combined operation, similar to the one dimensional string search and replace in word processors, proves very useful for detecting and modifying solid model descriptions with repetitive features. Our geometric search and replace operation is based on the solution of the following problem - “Given a labeled embedded graph G = (V, E, α) and a labeled pattern P = (C, ϕ), find all edges in E of G that are consistent with (C, ϕ)”. This problem is related to the subgraph isomorphism problem, but is much easier because of the given embeddings of the graph and the pattern. We present an efficient algorithm for this problem in which the total number of label comparisons required for any n vertex embedded graph G is no more than 4n, independent of the size of the pattern.

1 Introduction

Using geometric search one is able to find all matches of a certain pattern (a feature or a connected set of vertices, edges and faces) in the boundary representation of a solid model. Geometric replace is then the technique of replacing these matches with another suitable pattern (a feature or a connected set of vertices, edges and faces) which topologically agrees with the initial pattern. Coupled together, geometric search and replace techniques works in a manner similar to textual search and replace in word processors.

Our geometric search and replace operation is based on a solution of the following problem - “Given a labeled embedded graph G = (V, E, α) and a labeled pattern P = (C, ϕ), find all edges

*Supported in part by NSF grants CCR 90-02228, DMS 91-01424 and AFOSR contract 91-0276
†Supported in part by
2 THE MATCHING PROBLEM

in $\tilde{E}$ of $G$ that are consistent with $(C, \varphi)$". This problem is related to the subgraph isomorphism problem (which is NP complete [4]), but is computationally easier because of the given embeddings of the graph and the pattern. We present an efficient algorithm for this problem in which the total number of label comparisons required for any $n$ vertex embedded graph $G$ is no more than $4n$, independent of the size of the pattern.

Two dimensional editing using graphical search and replace has been implemented in several editing utilities. Using a toolkit called MatchTool [6] one could search for all objects matching graphical attributes such as fill color or line style and replace it a different set of attributes. However there is no way to search and replace geometric relationships or do complete shape-based replacements. Other two dimensional systems use automatic or interactive constraint generation based search and replace [9, 7, 8]. The illustrator of [9] searches for certain relationships such as nearly aligned lines or near coincident vertices and enforces these relationships precisely. In [7] user defined constraint rules can be graphically while [8] also allows a form of program input of two dimensional constraints. Graph search techniques for recognizing and matching features (convex edges, slots, through holes etc.) in solid models has been considered in several papers, see for e.g. [3, 5, 10]. In all these papers the approach has been a reduction to the subgraph isomorphism problem. Hence the solution techniques presented are based on clever polynomial time heuristics.

The rest of this paper is as follows. In section 2 we introduce the three dimensional embedded graph matching problem in more detail. In section 3 we present our efficient algorithm and a worst-case comparison analysis. Finally in section 4 we discuss the implementation of the geometric search and replace operation in our SHILP solid modeling toolkit [1].

2 The Matching Problem

An undirected graph $(V, E)$ having vertex set $V$ and edge set $E$ is called an *embedded graph* if it is mapped to an orientable 2-manifold in $\mathcal{R}$ in such a way that vertices are mapped to distinct points and edges are mapped to arcs connecting the two terminal vertices, and that edges do not have a point of intersection except at the vertices. An orientable 2-manifold divides the three-dimensional space into two connected components, bounded and unbounded; the former is called the *inside* and the latter the *outside*. Throughout the paper, we assume that the embedded graph is always seen from the outside of the 2-manifold, so that at each vertex we can uniquely specify the counterclockwise order of the edges incident to that vertex.

For an embedded graph $G = (V, E)$, we define set $\tilde{E}$ of directed edges by

$$\tilde{E} = \{(u, v), (v, u) \mid \{u, v\} \in E\},$$

and call the directed graph $(V, \tilde{E})$ the *parallelized graph* induced from $(V, E)$. We define two mappings $g_R$ and $g_L$ from $\tilde{E}$ to itself: for any $e = (u, v)$ in $\tilde{E}$, $g_R(e)$ is the directed edge going out of $v$ (other than $(v, u)$) that is first encountered when one moves counterclockwise around $v$, and $g_L(e)$ is the directed edge going out of $v$ (other than $(v, u)$) that is first encountered when one moves clockwise around $v$. Since $g_R$ and $g_L$ are one-to-one mappings from $\tilde{E}$ to itself, the inverses $g_R^{-1}$ and $g_L^{-1}$ are also one-to-one mappings from $\tilde{E}$ to itself. See Figure 1.

Let us consider the parallelized graph $(V, \tilde{E})$ as a network of one-way streets, and imagine a driver who drives a car in such a way that his car always faces in the direction specified by the edge and he can drive either forward or backward with the restriction that he should take the rightmost
2 THE MATCHING PROBLEM

FIGURE 1: A Parallelized Graph

FIGURE 2: Path Choices in a Parallelized Graph
Figure 3: A Labeled Pattern and part of a Labeled Parallelized Graph
turn or the leftmost turn at each vertex. Hence if the driver is at edge \( e \in \bar{E} \), the next edge he can visit is \( g_R(e), g_L(e), g_R^{-1}(e) \) or \( g_L^{-1}(e) \).

Let \( R, L, R^{-1} \) and \( L^{-1} \) denote his choice of \( g_R(e), g_L(e), g_R^{-1}(e) \) and \( g_L^{-1}(e) \), respectively, as the next edge, and let any concatenation of these four symbols denote the sequence of choice of the next edges with the convention that the choice is done from right to left. Hence, for example, \( LRL \) implies that the driver goes forward, takes the rightmost turn, takes the leftmost turn, and takes the leftmost turn again. So, if he starts at \( e_1 \) in Figure 2, he visits \( e_1, e_2, e_3, e_4 \) in this order.

Let \( R, L, R^{-1} \) and \( L^{-1} \) denote his choice of \( g_R(e), g_L(e), g_R^{-1}(e) \) and \( g_L^{-1}(e) \), respectively, as the next edge, and let any concatenation of these four symbols denote the sequence of choice of the next edges with the convention that the choice is done from right to left. Hence, for example, \( LLR \) implies that the driver goes forward, takes the rightmost turn, takes the leftmost turn, and takes the leftmost turn again. So, if he starts at \( e_1 \) in Figure 2, he visits \( e_1, e_2, e_3, e_4 \) in this order.

We call a finite sequence \( X_i X_{i-1} \cdots X_2 X_1 \) (\( X_j \in \{ R, L, R^{-1}, L^{-1} \}, j = 1, \ldots, i \)) of the symbols \( R, L, R^{-1}, L^{-1} \) a primary path choice. Note that a primary path choice is defined independently from the underlying graph. When we apply a primary path choice to a particular parallelized graph with a particular start edge, we get a sequence of edges of the graph. For primary path choice \( X_i \cdots X_1 \) and edge \( e \) of a parallelized graph, let \( X_i \cdots X_1(e) \) denote the edge that the driver reaches at the end of his move. Let \( \epsilon \) represent the primary path, choice of length 0, and we define \( \epsilon(e) = e \) for any edge \( e \). We define \( (R)^{-1} = R^{-1}, (L)^{-1} = L^{-1}, (R^{-1})^{-1} = R, (L^{-1})^{-1} = L \). Moreover, for \( b = X_i X_{i-1} \cdots X_2 X_1 \), we define \( b^{-1} \) by

\[
b^{-1} = (X_i X_{i-1} \cdots X_2 X_1)^{-1} = X_1^{-1} X_2^{-1} \cdots X_{i-1}^{-1} X_i^{-1};
\]

\( b^{-1} \) represents a primary path choice that is the reversal of \( b \), i.e., we can easily see that if \( e' = b(e) \), then \( \epsilon = b^{-1}(e') \). We define \( RR^{-1} \equiv R^{-1} R \equiv LL^{-1} \equiv L^{-1} L \equiv \epsilon \). The relation \( b \equiv b' \) represents that the two primary path choices \( b \) and \( b' \) give the same edge at the end of the moves along the paths. We call a primary path choice reducible if it can be replaced by a shorter primary path choice by the relation \( \equiv \), and irreducible otherwise.

Suppose that \( (V, \bar{E}) \) is a parallelized graph. Let \( \alpha \) be a mapping from \( \bar{E} \) to set \( A \), called a label set. For each \( e \in \bar{E} \), \( \alpha(e) \) is called the label of \( e \), and the triple \( (V, \bar{E}, \alpha) \) is called a labeled parallelized graph.

Let \( C \) be a collection of irreducible primary path choices. Let \( B(C) \) denote the set of all right substrings of strings in \( C \), that is,

\[
B(C) = \{ X_j X_{j-1} \cdots X_1 | X_i X_{i-1} \cdots X_1 \in C, 0 \leq j \leq i \}.
\]

Hence, in particular, \( B(C) \) always contains the null string \( \epsilon \). An element of \( B(C) \) itself is an irreducible primary path choice. An element of \( B(C) \) can be considered as the representation of an edge which the driver can reach when he drives according to some primary path choice in \( C \). In particular, \( \epsilon \) represents the start edge. Let \( \varphi \) be a mapping from \( B(C) \) to \( A \). For \( X_j \cdots X_1 \in B(C) \), \( \varphi(X_j \cdots X_1) \) intuitively represents the label of the terminal edge of the primary path choice \( X_j \cdots X_1 \). We call the pair \( (C, \varphi) \) a labeled pattern.

An edge \( e \in \bar{E} \) is said to be consistent with primary path choice \( X_j \cdots X_1 \) in \( B(C) \), if \( \varphi(X_j \cdots X_1) = \alpha(X_j \cdots X_1(e)) \). An edge \( e \) is said to be consistent with the labeled pattern \( (C, \varphi) \) if \( e \) is consistent with all primary path choices in \( B(C) \).

**Problem 2.1** Given a labeled parallelized graph \( (V, \bar{E}, \alpha) \) and a labeled pattern \( (C, \varphi) \), find all edges in \( \bar{E} \) that are consistent with \( (C, \varphi) \).
This problem is related to the subgraph isomorphism problem but is not the same. The difference can be understood in the following example. Let $C = \{RR, LR, R^{-1}\}$. Then, we get $B(C) = \{\epsilon, R, RR, LR, R^{-1}\}$. Let $\varphi$ be a map such that $\varphi(\epsilon) = a$, $\varphi(R) = b$, $\varphi(RR) = c$, $\varphi(LR) = a$, $\varphi(R^{-1}) = c$. Then, the labeled pattern $(C, \varphi)$ can be represented by the labeled tree structure shown in Figure 3(a), where the directed edge $e$ represents the start edge, a small arc connecting two edges represents the relation that the associated edges are immediate neighbor of each other in the cyclic list of edges around the vertex, and the symbols in the parentheses represent the labels defined by $\varphi$. Next, let Figure 3(b) be a part of a labeled parallelized graph with labels represented by symbols in parentheses, in which one of each pair of parallelized edges is omitted. We can easily see that edge $e_1$ in (b) is consistent with the labeled pattern $(C, \varphi)$. Actually, this gives a subgraph isomorphism. However, edge $e_2$ in (b) is also consistent with the labeled pattern $(C, \varphi)$, though the corresponding edges in (b) form a cycle. Moreover, edge $e_3$ in (b) is also consistent with $(c, \varphi)$; in this case two edges in (a), i.e., the edges associated with $RR$ and $R^{-1}$, correspond to the same edge in (b). Thus, the solution of Problem 1 gives a wider class of matching than the class of subgraph isomorphisms.

3 Algorithmic Details

We consider the next algorithm for solving Problem 1. In the algorithm, we use two arrays $d(e, j)$ and $h(e)$, where the argument $e$ runs in $\widetilde{E}$ and the argument $j$ runs in $\{1, 2, \ldots, k\}$. The value of $d(e, j)$ is "unknown", "match" or "mismatch", where "match" means that the edge $e$ is consistent with the $j$th primary path choice in $B(C)$, and "mismatch" means that the edge $e$ is not consistent with the $j$th primary path choice in $B(C)$. The value of $h(e)$ is "unknown", "consistent" or "inconsistent"; "consistent" means that $e$ is consistent with the pattern label $(C, \varphi)$ whereas "inconsistent" means that $e$ is not consistent with $(C, \varphi)$. The two lines in brackets in the algorithm are not necessary for the actual algorithm, but are useful for the later discussion of the behavior of the algorithm.

Algorithm 1.
Input: a labeled parallelized graph $(\widetilde{V}, \widetilde{E}, \alpha)$ and a labeled pattern $(C, \varphi)$.
Output: all the edges in $\widetilde{E}$ that are consistent with $(C, \varphi)$.
Preprocessing:
1. Assign a linear order, say $b_1, b_2, \ldots, b_k$, to the elements of $B(C)$ in such a way that $b_1 = \epsilon$.
2. For each $i = 1, 2, \ldots, k$, create two sets:
   
   $$S_i = \{(b_j^{-1}b_i, j) \mid b_j \in B(C), \varphi(b_j) = \varphi(b_i)\},$$
   $$T_i = \{(b_j^{-1}b_i, j) \mid b_j \in B(C), \varphi(b_j) \neq \varphi(b_i)\}.$$

Main processing:
1. $d(e, j) \leftarrow "unknown"$ for all $e \in \widetilde{E}$ and for all $j = 1, \ldots, k$.
2. $h(e) \leftarrow "unknown"$ for all $e \in \widetilde{E}$.
3. while there exists element $e \in \widetilde{E}$ having $h(e) = "unknown"$, choose such an element $e$ and do
   
   $\begin{align*}
   i & \leftarrow 1;
   \end{align*}$

LOOP:
if \( d(e,i) = \text{"unknown"} \) then
if \( \alpha(b_i(e)) = \varphi(b_i) \) then
\( d(b_j^{-1}b_i(e), j) \leftarrow \text{"match" for each } (b_j^{-1}b_i, j) \in S_i; \)
\( h(b_j^{-1}b_i(e)) \leftarrow \text{"inconsistent" for each } (b_j^{-1}b_i, j) \in T_i; \)
\[ d(b_j^{-1}b_i(e), j) \leftarrow \text{"mismatch" for each } (b_j^{-1}b_i, j) \in T_i; \]
else
\( h(b_j^{-1}b_i(e)) \leftarrow \text{"inconsistent" for each } (b_j^{-1}b_i, j) \in S_i; \)
\[ d(b_j^{-1}b_i(e), j) \leftarrow \text{"mismatch" for each } (b_j^{-1}b_i, j) \in T_i; \]
goto NEXT;
endif
endif
i \leftarrow i + 1;
if \( i \leq k \) then goto LOOP else \( h(e) \leftarrow \text{"consistent"} \) endif;
NEXT:
end

Lemma 3.1 \( S_i \) is nonempty and \( |S_i \cup T_i| = k \) for \( i = 1, 2, \ldots, k \). Moreover, \( S_1, \ldots, S_k, T_1, \ldots, T_k \) are mutually disjoint.

Proof: \( S_i \) contains \( (\epsilon, i) \) and hence nonempty. From the definition, \( S_i \) and \( T_i \) are disjoint and \( |S_i| + |T_i| = k \) for \( i = 1, 2, \ldots, k \). Suppose that \( S_i \cup T_i \) and \( S_i \cup T_l \) have the same element \( (b_j^{-1}b, j) \). Then, \( b \) must satisfy \( b = b_i = b_l \), which means \( i = l \). Thus \( S_1, \ldots, S_k, T_1, \ldots, T_k \) are mutually disjoint.

Lemma 3.2 Algorithm 1 puts a value to each entry of the array \( d(e, j) \) at most twice, once the value “unknown” and the other time either “match” or “mismatch”.

Proof: Suppose, against the proposition, that the value “match” is put in \( d(e, j) \) twice, once at the time when we get \( \alpha(b_l(e')) = \varphi(b_l) \) (i.e., when we come to know by a label comparison that edge \( e \) is consistent with the primary path choice \( b_i \)) and once more at the time when we get \( \alpha(b_l(e'')) = \varphi(b_l) \). This in particular implies that (i) \( \varphi(b_j) = \varphi(b_l) = \varphi(b_l) \), and (ii) \( e = b_j^{-1}b_l(e') = b_j^{-1}b_l(e'') \). From (i) and the definition of \( S_i \), we get (iii) \( S_i \ni (b_j^{-1}b_l, i) \). From (ii) we get (iv) \( e'' = b_j^{-1}b_l(e') \). The two facts (iii) and (iv) together imply that when we get \( \alpha(b_l(e')) = \varphi(b_l) \) by a label comparison, Algorithm 1 should put “match” to \( d(e'', l) \). Hence the label comparison to see whether \( \alpha(b_l(e')) = \varphi(b_l) \) will never been done, which contradicts our assumption. We get similar contradiction if we assume that the value “mismatch” is put in \( d(e, j) \) twice.

Lemma 3.3 In Algorithm 1 the label comparisons (i.e., the check to see whether \( \alpha(b_i(e)) = \varphi(b_i) \) is satisfied) are done at most \( 4n \) times, where \( n = |\bar{E}|/2 \) (i.e., \( n \) is the number of edges of the original graph from which the parallelized graph is created).

Proof: Suppose that the two procedures in the brackets are also done. The algorithm terminates when each edge \( e \in \bar{E} \) has either \( h(e) = \text{"consistent"} \) or \( h(e) = \text{"inconsistent"} \). This implies that the algorithm terminates at latest when all of \( d(e,i) \) have values other than “unknown”. Let \( l \) denote the number of label comparisons that result in “true”, and \( m \) denote the number of label
comparisons that result in "false". From Proposition 1, the values of entries of the array $d(e, i)$ change at least $kl + m$ times, and from Proposition 2 the same entry of the array is not changed more than once. The size of the array is $2kn$, and hence we get $kl + m \leq 2kn$. Moreover, we get $m \leq 2n$ because if the label comparison results in "false", we immediately go to the next edge. Thus, the maximum number of label comparisons is not greater than the solution of the maximization problem: "maximize $l + m$ subject to $kl + m \leq 2kn$ and $m \leq 2n"$, and consequently we get $\max(l + m) < 4n$.

4 Applications to Solid Model Editing

Geometric search and replace has a number of applications in the editing of solid model descriptions. It can be used to make multiple changes to a model or changes to many models at once. If the replacement pattern is more complex than the search pattern, the multiple changes can turn a simple solid model into an elaborate one. Furthermore, if the replacement pattern contains the search pattern as an embedded subgraph, then the geometric search and replace can be applied recursively.

![Figure 4: Using a Vertex Pattern Search and Replace to Bevel a Cube](image)
We have implemented our geometric matching algorithm in our X-11 based interactive solid modeling and display toolkit called SHILP [1]. The user can interactively select a solid model (or a group of solid models) and input both a search and a replace pattern. The search and replace patterns can be input by mousing in a pattern or by selecting features (connected embedded subgraph of vertices, edges or faces) of any resident solid model. For example, as shown in Figure 4 the search pattern is the subgraph of three edges and is selected from a corner of the cube model itself. The replace pattern is a triangle and is moused in on the search pattern. This relatively referencing of the replace pattern with respect to the search pattern allows the user to input a topologically consistent replacement embedded subgraph. The result of the geometric search and replace for this example is a beveling of all the corners of the cube. Similar repetitions of replacement of a selected corner type by a triangle, as shown in Figure 5, automatically yields multiple matches and replacements for elaborate resulting models starting from a simple cube.

Our current implementation allows arbitrary search and replace patterns to be input by the user. Arbitrary replacement patterns, of course, do not make sense in the consistent editing of solid models, as the topology of the modified solid may be inconsistent (dangling edges, or faces or may no longer enclose finite volume, etc..). The problem of determining which replacement patterns yield consistent modified solid models is quite challenging. The problem is complicated by the fact that replacement patterns may often interact and at times in subtle ways, yet yield consistent final topologies. See for example Figure 6 where the search pattern is an indegree five corner of
an icosahedron, and the replacement pattern is a beveling of both the vertices and edges and is referenced off of the search pattern. In this case our geometric search and replace operation detects the interaction and sequences the search and replacements to yield a consistent rounded shape of the input icosahedron model.

5 Extensions

Several extensions are possible in both our algorithm for geometric pattern matching as well as in harnessing its power for solid model editing. For now the matching only applies to rigid polyedral solid models. One possible extension is to allow geometric search and replace operations on solid models with algebraic curves and surface boundaries [2]. We are currently upgrading our geometric search and replace operation in SHILP to manipulate such curved surface models, by comparing curve and surface equations along with the other geometric relationships.

Another important extension is to allow tolerances in the search patterns which can be achieved by labeling small ranges for the geometric features of the embedded subgraph. A third very desirable is an inbuilt consitency checker for user input search and replace patterns. Finally, a powerful extension would be to enhance the allowable constraints in our current pattern matching. For now

Figure 6: Interacting Vertex and Edge Replacement Patterns with Consistent Topology
the constraints are those which arise directly from the embedded property of the search pattern. Variational constraints for example, may allow specific geometric relationships to be sought and maintained as a result of such constraint based geometric search and replace operations on a solid model.

Acknowledgement: We thank Dan Schikore for his help with the implementation of the matching algorithm.

References


