Approximate Topological Matching of Quadrilateral Mes

Abstract

We study approximate topological matching of quadrilateral meshes, that is, the problem of finding as large a set as possible of matching portions of two quadrilateral meshes. This study is motivated by applications in graphics that involve shape modeling whose results need to be merged in order to produce a final unified representation of an object. We show that the problem of producing a maximum approximate topological match of two quad meshes in NP-hard. Given this result, which makes an exact solution extremely unlikely, we show that the natural greedy algorithm derived from polynomial-time graph isomorphism can produce poor results, even when it is possible to find matches with only a few non-matching quads. Nevertheless, we provide a “lazy-greedy” algorithm that is guaranteed to find good matches when mis-matching portions of mesh are localized. Finally, we provide empirical evidence that this approach produces good matches between similar quad meshes.

Index Terms: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Geometrical problems and computations; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations

1 Introduction

Quadrilateral meshes represent polyhedral surfaces in such a way that each face is a quadrilateral, which is quite useful in computer graphics applications. The usefulness of these meshes comes in part from the fact that rectangular texture images are easily mapped without clipping into quadrilaterals (which we sometimes refer to as “quads”). Thus, quad meshes are desirable for surface modeling.

Quad meshes are also useful in finite element analysis, where one desires adaptive refinement into regions of interest, since quadrilaterals are easily partitioned into finer quadrilateral grids. Indeed, several mesh generation algorithms, such as that of Schonfeld and Weinerfelt (31) and others (1; 26; 34), begin with a coarse quadrilateral mesh and then refine each quad of interest into a regular grid of sub-quads.

In order to best utilize the time of people working with a given quad mesh, it is common for a single mesh to be used in a number of different processing paths in parallel. For example, a single mesh might be processed simultaneously for texture mapping, feature mapping, finite-element analysis for physical simulation purposes, and morphing for object animation. Unfortunately, the software systems that perform the multiple simultaneous tasks on a given mesh $M$ often use different internal representations of $M$, which in turn result in different representations of $M$ in the output of each task. For example, different mesh processing software systems could be developed by different vendors, each with its own proprietary way of storing of meshes.

More importantly, many of the computational tasks performed on meshes are likely to slightly change their structure in places. For example, a physical simulation or animation task may introduce a “rip” in a surface or a modeler may detach a model feature (such as a hand or eyebrow), modify it to reflect a new pose, and re-attach it to the original model. In addition, models that possess significant symmetries, such as biological characters and architectural models, often have their texturing and feature mapping tasks performed on a single portion with the desire that this work be mapped to the symmetric portion(s), even if there are a few localized asymmetries to deal with during this mapping process.

For the above applications, we are interested in matching pairs of quad meshes in a way that is as independent of geometric features as possible. That is, we would like matches that align the topological structure between two given quad meshes as best as possible, matching the orientations of each paired-up vertex and quad in the two meshes. Therefore, we are interested in this paper in the approximate topological matching problem, which takes two given quad meshes and finds the best matching between them based on topological information alone. (See Figure 1.)

Figure 1: The Approximate Topological Matching Problem. Quads in (dark) red denote mismatched regions. In spite of there being major pieces of the first mesh missing from the second, the parts of similarly are still matched (the two right hands actually don’t match topologically, as the hand of the second model is attached differently).

1.1 Prior Related Work

Before presenting our results, let us review prior related work on quad meshes and mesh matching.

*http://www.ics.uci.edu/~eppstein
†http://www.ics.uci.edu/~goodrich
‡ethan@cs.mcgill.ca
§Rasmus.Tamstorf@disney.com

University of California, Irvine
University of California, Irvine
McGill University
Walt Disney Animation Studios
1.1.1 Quad Mesh Generation

This is not a paper on mesh generation, but let us nevertheless mention some of the modeling approaches that give rise to quad meshes. There are several quad mesh generation methods that involve starting from a coarse quad mesh and subdividing each coarse quad into a finer structured mesh (1; 26; 31; 34). Of course, there are also methods that do not create structured submeshes as a refinement step (4; 6; 8; 28; 32). Although this is not a paper on mesh generation, we note that the mesh generation process often provides us with properties we can exploit in our algorithms. For example, many mesh generation algorithms create large patches of structured submeshes, each having degree 4. Such submeshes are still relatively common, of course, even if not directly constructed, since, by an easy argument that follows directly from Euler’s formula, the vertices in any quad mesh of bounded genus have average degree equal to 4.

1.1.2 Mesh Compression

One tool we use in our approximate topological matching algorithm is to compress the original quad mesh so as to reduce the number of candidate starting points for our matching process. The problem of mesh compression has historically been studied in the classic sense of data compression, where one wishes to produce a concise representation of a mesh for the sake of reduced transmission and storage costs (9; 18; 20; 21; 35; 36). Our reason for using mesh compression instead is based on the desire to speed up the computation time in an algorithm that operates on quad meshes. Thus, our approach actually fits the spirit of other algorithms (e.g., see (14; 15; 33; 38)) that perform data compression so as to improve algorithmic performance.

1.1.3 Graph Isomorphism

In the classic graph isomorphism problem, we are given a graph \( G = (V, E) \) and a graph \( H = (W, F) \), with \( |V| = |W| \) and \( |E| = |F| \), and we are asked if there is a mapping \( f : V \rightarrow W \) such that edge \( e = (v, w) \) is an edge in \( E \) if and only if \( (f(v), f(w)) \) is an edge in \( F \). Applied to the mesh matching problem, graph isomorphism corresponds to the problem of finding an exact topological match, with no edges, vertices, or faces unmatched between the two input meshes. Much work has been done on the graph isomorphism problem for general classes of graphs (e.g., see (10; 17; 24)), for which it is not known whether graph isomorphism can be solved in polynomial time or if the graph isomorphism problem is NP-complete. For the special case of planar graphs, or more generally for models of bounded genus, isomorphism can be solved in linear time (19; 23), but these algorithms are fairly complex.

The problem of solving graph isomorphism for meshes can be solved in polynomial time, even for models of high genus. The reason is that, in addition to their graph structure, meshes possess structure derived from the fact that they are embedded in manifold surfaces. In particular, such embeddings impose a specific, given ordering on the edges around each vertex, \( v \), which corresponds to a clockwise or counter-clockwise listing of the edges incident to \( v \). Thus, given a mapping of an edge in \( G \) to an edge in \( H \), it is a simple matter to “grow out” the respective corresponding matching between \( G \) and \( H \) so long as such matches are possible. That is, there is a simple \( O(m^2) \) algorithm for finding an exact topological match between two \( m \)-edge meshes (or determining that no such matching exists)—attempt to grow out a possible match between \( G \) and \( H \) starting from each possible matching of edges in \( G \) to a given edge in \( H \). In practical applications, such algorithms are still too slow, however, and we address practical algorithms for the graph isomorphism problem on quad meshes in a different paper (which is currently under submission elsewhere). Our approach in this other paper is based on a technique that is inappropriate for solving the approximate topological matching problem.

1.1.4 Graph Edit Distance

The approximate topological matching problem is related to the problem of computing the edit distance between two graphs (5; 7; 13; 27). The concept of graph edit distance was first introduced by Eshera and Fu (13). It involves the computation of a measure of similarity between two graphs based on the minimum number of edit operations, such as edge insertions/deletions and vertex insertions/deletions, that would be needed to convert one graph into another, which is a problem shown to be NP-hard. Subsequent work has focused on heuristic methods for computing graph edit distance. For example, Berretti et al. (5) describe a distance metric on graphs and apply it to content retrieval for color images, and Neuhaus and Bunke (27) use automatic learning methods to compute the functions for graph edit distance.

The approximate topological matching is not the same as graph edit distance, however. Graph edit distance is concerned with minimizing the number of edit operations to convert one graph to another, whereas approximate topological matching is concerned with maximizing the amount of common matching quads between two quad meshes, even if there are large portions of one of the two meshes that might not match with any part of the other. Thus, the approximate topological matching problem is an adaptation of the maximum common subgraph problem (22) to quad meshes, where we wish to match oriented quad faces as well as vertices and edges. The distinction between edit distance and approximate topological matching is important in object modeling applications, for instance, where one mesh could be an early version of a model, such as a character’s face, before new facial features are added, such as eyebrows, eye lashes, and warts, and we want to match the earlier version as best as possible against the current version. In such cases it is better to ignore the mismatching parts rather than weigh them negatively in an edit distance value.

1.2 Our Results

In this paper, we study the approximate topological matching problem for quadrilateral meshes. We show that this problem is NP-hard, implying that it is very unlikely that there is a polynomial-time algorithm for approximate topological matching. Thus, we develop a heuristic algorithm for approximate topological mesh matching, based on the use of a skeleton graph that is a well-defined, robust subgraph of the mesh. This skeleton is then used to identify anchors to begin a matching process similar to the way the medial axis is visualized for a polygon (e.g., see (11; 29)). If we imagine that the plane is made of a combustible material and we start a fire starting from each boundary edge of the polygon, then the medial axis is defined by the places where two waves of fire meet. In our case, we apply this approach to quad meshes, viewing the quads as “combustible material” and the anchors as the starting points for our fires. We grow regions in the two input meshes simultaneously, matching similar topological substructures as we go.

Such a fire-propagation approach to matching the two input meshes has an unfortunate complication, however. Namely, if we use the natural greedy approach to propagate matching portions of the wavefront, then this process can lead to poor matches. In particular, we show that even if one small portion of one mesh is shrunk relative to a portion in the other, this greedy algorithm can produce globally poor matches. Nevertheless, we show that our fire-growing approach to approximate matching can be made to work effectively even in these cases, by growing the wavefronts in a “lazy-greedy” fashion. In addition, we provide experimental results that show that the lazy-greedy algorithm can produce good matches quickly in practice.

2 Preliminaries

Before we discuss our results, let us give some preliminary definitions and observations about quadrilateral meshes.
2.1 The Topology of Quad Meshes

Any graph drawn on the sphere $S_0$ in three-dimensional space partitions the surface of $S_0$ into cells such that each is homeomorphic to a disk (we will always assume in this paper that graphs are drawn without edge crossings). Each such cell is called a face in the embedding. Adding $g$ “handles” to $S_0$ gives the surface $S_g$, which is said to have genus $g$. For example, a traditional coffee cup is of genus 1. Likewise, the dog toy shown in Figure 2 has genus 16.

![Figure 2: A dog toy, which has genus 16.](image)

A cellular embedding of a graph on $S_g$ is a drawing that partitions $S_g$ into cells such that each is homeomorphic to a disk. We define a quadrilateral mesh to be a cellular embedding of a graph $G = (V,E)$ onto a surface $S_g$ such that each face is a quadrilateral. We say that the genus of the mesh is $g$ in this case. We therefore view a quad mesh as a triple $(V,E,Q)$ where $V$ is a set of vertices, $E$ is a set of edges, and $Q$ is a set of quadrilaterals. In addition, we assume that quad meshes are represented with a data structure, like the “winged edge” structure (3), that supports the following operations:

- List the incident edges around a given vertex (in clockwise or counter-clockwise order) in time proportional to the degree of that vertex.
- List the bounding edges around a given face (in clockwise or counter-clockwise order) in time proportional to the size of that face.
- List the two vertices that are the endpoints of a given edge in constant time.

The simplest form of quad mesh is a structured mesh, where every vertex has degree four. The average degree of vertices in a bounded-genus quad mesh is 4, and it is common for the majority of vertices in a quad mesh to have degree 4. For this reason, if a vertex in a quad mesh is an interior vertex with degree different than 4, or an exterior (boundary) vertex with degree different than 3, then we refer to that vertex as an extraordinary vertex.

2.2 The Size of a Quad Mesh

Traditionally, an algorithm operating on a graph $G$ is characterized in terms of $n = |V|$, the number of vertices of $G$, and $m = |E|$, the number of edges in $G$. We have these measures in quad mesh algorithms as well, but we also have $q = |Q|$, the number of quads, so it is useful to relate these quantities.

**Observation 1** In a quad mesh with $m$ edges and $q$ faces, $2q \leq m \leq 4q$.

**Proof:** If we sum up the number of edges on every face we will count each edge at least once and at most twice. Since each face is a quadrilateral, $4q \leq 2m$ and $m \leq 4q$.

**Observation 2** The number of edges and faces in a quad mesh can be arbitrarily larger than the number of vertices.

**Proof:** We can place two vertices at opposite poles of a sphere and add as many edges as we like joining them. Now remove the sphere and enlarge each edge to be a thin tube. Next, take each original vertex and split into two points separated by the width of a tube, with one on top and one on the bottom. This creates a shape similar to the dog toy shown in Figure 2. Now, for each tube, run an edge between the two top vertices, an edge between the two bottom vertices, and an edge joining top to bottom at each pole. This creates a quad mesh with four vertices and an arbitrary number of faces and edges.

Although it is not uncommon in the solid modeling literature to allow for such multiple edges and even self loops in the graph defined by a quadrilateral mesh, we will restrict ourselves in this paper to simple meshes, where there disallow multiple edges between the same pair of vertices and we disallow self loops. Likewise, we disallow multiple edges and self loops in the dual graph, which is formed by placing a vertex in each quad and joining two quads $Q$ and $R$ with an edge any time $Q$ and $R$ have an edge of the mesh in common. Likewise, we require that the mesh be well-formed, meaning that it satisfy the following:

1. For each vertex $v$, the set of quads containing $v$ is connected in the dual graph.
2. The boundary of a quadrilateral mesh $M$ consists of all edges of $E$ that belong to exactly one quadrilateral in $M$ and all vertices incident to an edge of this type. For every cyclic portion $C$ of the boundary of $M$ on $S_g$ (that is, a “hole” in the mesh), the interior of $C$ is homeomorphic to a disk. That is, each hole in $M$ is contractible.
3. Every edge in $M$ is adjacent to at least one and at most two quadrilateral faces of $M$.
4. Any two quadrilaterals in $Q$ intersect in a single edge, a single vertex, or the empty set.

**Observation 3** In a simple, connected, well-formed quadrilateral mesh $M$ of genus $g$, with $n$ vertices and $m$ edges, 

$$m \leq 2n + 4g - 4.$$ 

**Proof:** Since $M$ is a simple, cellularly-embedded graph in $S_g$, the Euler characteristic implies that

$$n - m + q = 2 - 2g.$$ 

Specifically, the Euler characteristic (which is also called “Euler’s formula” or the “Euler-Poincaré characteristic”) states that, in such embeddings, the number of vertices minus the number of edges plus the number of faces is equal to $2 - 2g$. That is, in our case, $m = n + q + 2g - 2$. By Observation 1, $q \leq m/2$. Thus, $m \leq 2n + 4g - 4$.

In almost all practical applications of quadrilateral meshes, the genus $g$ of a given mesh is bounded by constant and it is almost certainly $O(n)$. In fact, $g$ is typically 0 or 1. Thus, the above lemma implies that in almost every practical application using a mesh $M$ with $n$ vertices and $m$ edges, $m$ is $O(n)$. Thus, for example, the time complexity of the simple wave-growing exact graph isomorphism algorithm on such meshes is $O(n^2)$. We also have the following.

**Lemma 4** In a simple, connected, well-formed quadrilateral mesh $M$ of genus $g$, with $n$ vertices and $m$ edges,

$$m \leq \left(\frac{4}{3}g + 2\right)n - 4.$$
Proof: The proof follows that of a similar lemma of Wood and Telle (37), but is adapted to quad meshes. From Observation 3, we know that
\[ m \leq 2n + 4g - 4. \]
So we need to show that \(Ag \leq \left(\sqrt{\frac{n}{2}}\right)n.\) That is, we need to show \(g \leq n^2/12.\) This follows from the fact that \(M\) is simple and the complete graph on \(n\) vertices, \(K_n,\) has genus at most \(n^2/12\) (e.g., see (25)). \(\blacksquare\)

3 On the Difficulty of Approximate Topological Matching

As mentioned above, the problem of finding a best approximate topological match between two quad meshes has several uses in object representation and rendering applications. Unfortunately, as we show in this section, the problem of finding an optimal approximate topological match is NP-hard.

In order to be precise, let us formalize the approximate topological matching problem for quad meshes. Suppose we are given two quadrilateral meshes, \(M_1\) and \(M_2.\) A matching submesh \(S_1\) of \(M_1,\) with respect to \(M_2,\) is a connected set of quads in \(M_1\) that is mapped one-to-one to a connected set of quads in \(M_2\) by a function \(\mu\) that maps quads in \(S_1\) to quads in \(M_2\) such that \(q_1\) and \(q_2\) are adjacent in \(S_1\) if and only if \(\mu(q_1)\) and \(\mu(q_2)\) are adjacent in \(M_2\) (using adjacency across edge boundaries). The approximate topological matching problem is to find a matching submesh \(S_1\) of \(M_1,\) with respect to \(M_2,\) and corresponding mapping function \(\mu,\) such that \(S_1\) has the largest number of quads over all such submeshes. Unfortunately, we have the following.

Theorem 5 The approximate topological matching problem for quad meshes is NP-hard.

Proof: We will give a reduction from the known NP-complete problem of determining whether a given cubic planar graph is Hamiltonian (16). Suppose then that we are given an \(n\)-vertex cubic planar graph \(G\) as input, that is, a graph \(G\) that has degree 3 and which can be drawn in the plane without crossings. Our proof is based on showing that we can construct two meshes \(M_1\) and \(M_2\) in polynomial time such that all but \(n\) quads of \(M_1\) can be matched with part of \(M_2\) if and only if \(G\) is Hamiltonian, that is, \(G\) contains a cycle that visits each vertex in \(G\) exactly once. We begin our construction by using any existing polynomial-time method (e.g., see (2; 12; 30)) to produce an embedding of \(G\) in an \(O(n) \times O(n)\) integer grid, where \(n\) is the number of vertices in \(G.\) This embedding allows us to associate integer coordinates in the plane to the vertices of \(G,\) from which we will build a quad mesh, \(M_2.\) Before we perform this construction, however, let us first build the mesh \(M_1\) that we want to match completely to \(M_2.\)

In particular, let \(C\) be a simple cycle with \(n\) vertices, say, embedded at regular intervals around the boundary of a sufficiently large circle in the plane (separate from the grid \(G\) is embedded in). We construct a mesh \(M_1\) from \(C\) by expanding each edge of \(C\) into a connected sequence of four quads, linked in a “chain,” and expanding each vertex of \(C\) according to the replacement submesh of four quads shown in Figure 3a. That is, if we have an edge \((u, v)\) in \(C\) connecting \(u\) to \(v\) that is followed by an edge \((v, w)\), we replace \((u, v)\) by a chain of four quads that connect to \(v\)’s four quads as in the lower-right part of Figure 3a and we have the chain of four quads for \((v, w)\) connect as in the upper-left part of Figure 3a.

We convert \(G\) into a second quad mesh \(M_2\) by expanding each edge into a chain of four quads, as in our construction of \(M_1,\) but we connect them to vertices in a different way. Specifically, each vertex in \(G\) has degree 3, so we instead connect each edge chain of four quads according to the replacement mesh shown in Figure 3b.

The dashed lines in the figure show all the possible ways that a maximum number of quads in a vertex submesh from \(M_1\) can match a maximum number of quads in the vertex submesh in \(M_2.\) Namely, a vertex submesh of 4 quads in \(M_1\) can match at most 3 quads in a vertex submesh \(M_2.\) That is, all but one of the quads in a vertex submesh of \(M_1\) can match in a vertex submesh of \(M_2.\) Also, note that the same cannot be said of the submeshes of \(M_1\) associated with edges of \(C.\) Namely, note that since each edge in \(C\) is expanded into a connected chain of 4 quads, the quads in an edge submesh of \(M_1\) can match at most 2 quads in a vertex submesh of \(M_2.\) In other words, to get the largest number of matching quads between \(M_1\) and \(M_2,\) we need to match vertex submeshes in \(M_1\) to vertex submeshes in \(M_2.\)

So we have yet to show that \(G\) is Hamiltonian if and only if all but \(n\) of the quads in \(M_1\) can be matched with quads in \(M_2.\) Suppose that \(G\) is Hamiltonian. That is, \(G\) is a subgraph of \(G;\) hence, we can match each vertex submesh of \(M_1\) with a vertex submesh of \(M_2,\) using the cycle \(C\) as a guide on how to similarly match each edge submesh of \(M_1\) with an edge submesh of \(M_2,\) again, using the embedding of \(C\) in \(G\) as a guide. Then there is a topological match of \(M_1\) in \(M_2\) that pairs up all but \(n\) quads from \(M_1,\) i.e., the number of matched quads between \(M_1\) and \(M_2\) is \(7n.\)

Suppose, on the other hand, that there is a match of \(M_1\) and \(M_2\) that matches \(7n\) quads, that is, all but \(n\) quads from \(M_1.\) As we have noted above, the only way this can occur is if the edge submeshes of \(M_1\) match edge submeshes of \(M_2\) and 3 quads in each vertex submesh of \(M_1\) match inside a vertex submesh of \(M_2.\) Thus, \(C\) is a subgraph of \(G,\) that is, \(G\) is Hamiltonian. This completes the proof. \(\blacksquare\)
4 Our Algorithm

Given that the approximate topological matching problem is NP-hard, it is very unlikely that there is an efficient algorithm for solving it exactly. Thus, let us discuss a heuristic approach.

4.1 Identifying Anchors

Let us begin with the starting point for our heuristic algorithm—the identification of good anchors that can seed the process of matching pairs of quads in the input meshes $M_1$ and $M_2$. That is, extraordinary vertices in $M_1$ and $M_2$ that have relatively distinctive neighborhoods. Note: if there are no extraordinary vertices at all (e.g., if $M_1$ and $M_2$ are tori), then we pick an arbitrary pair of vertices in $M_1$ and $M_2$ as anchors.

In order to find a good set of anchors in the general case, we apply a few iterations of the Weisfeiler and Leman (WL) algorithm for exact graph isomorphism (e.g., see (17)). Recall that in the WL algorithm, we initially label each vertex with a label associated with its degree. Then, each vertex is labeled by a string of its neighbors’ labels in an order that appears in the topological embedding. In turning the cyclic ordering into a linear ordering, we pick the one that is lexicographically minimum. Then we let these strings be the new labels of the vertices (which we can re-normalize to be the integers from 1 to $n$ with a simple radix sort). Algorithm 1 gives a pseudocode for this procedure, which views the vertex labels as “colors.”

```
foreach vertex $u \in M$ do
    tmpColor[$u$] = deg($u$);
end
foreach $u \in M$ do
    Color[$u$] = ();
    foreach $v \in neighbor(u)$ do
        Color[$u$] = Color[$u$] + tmpColor[$v$];
    end
    Color[$u$] = Lexicographical minimum ordering of Color[$u$];
end
return the set of vertices with unique color label;
```

Algorithm 1: Algorithm for ColorGraph

Observe that the second loop in Algorithm 1 can be repeated to refine the color labels of vertices. This repetition should be done until a reasonable stopping condition is reached. For example, we used the stopping condition of repeating until we reach a set upper bound, $k$, on the number of iterations or until at least one pair of uniquely labeled vertices are found, one from each mesh. If we repeat this loop $i$ times, the label of each vertex $u$ will contain information about the vertices that are within distance $i$ from $u$. The running time of this second loop is $2m = O(m)$, which is $O(n)$ in the case of planar meshes or meshes of at most linear genus, by Observation 3.

The final step of the algorithm for identifying seeds to initiate mesh-matching growth from is that of finding corresponding anchors from the labeled compressed meshes. In order to find such seeds, we first match the rarest (hopefully unique) labeled vertices returned from the two compressed meshes, and then look for matching neighbors of the two matched vertices. Again, for most meshes, this process takes $O(n)$ time in the worst case, by Observation 3 or Lemma 4.

1 Not only the degree of each vertex can serve as the seed for labels, we may also choose to use the edge weights in the compressed meshes. In order to do so, we can label each edge by number of edges contracted while compressing the meshes.

4.2 Skeleton Graphs

As noted above, we are interested in finding rare or even uniquely-labeled anchors, so as to limit the number of possible candidate starting points for growing matching sets of quads between the two input meshes. Thus, it makes intuitive sense that we should concentrate on extraordinary vertices. In order to focus on the adjacencies between these vertices, we apply a compression scheme that focuses on extraordinary vertices and in most cases reduces the size of the graph we must deal with.

4.3 Particle Shooting

The idea for constructing this compressed skeleton graph inside each mesh is quite simple: we imagine that we shoot a particle out along every possible edge going out of each extraordinary vertex. These particles travel separately along the edges going out from extraordinary vertices. When a particle enters a normal vertex it continues through that vertex and leaves out the opposite side (unless this is a boundary vertex and there is no edge on the other side). In propagating these fictitious particles in this way we trace out a subgraph in each input mesh, $M_1$ and $M_2$. We let the particles continue to move, tracing out the compressed graph we are going to use for color label assignment, until each such particle reaches another extraordinary vertex (which is a very common occurrence given the way that people build quad meshes in practice) or the particle reaches a boundary edge. We then perform our anchor-finding procedure on this skeleton graph. (See Figure 4.)

![Figure 4: Anchor finding in a simple quad mesh, $M_1$. This mesh has five extraordinary vertices, consisting of the four corners and the vertex marked "start." The vertex marked "start" is the best candidate for an anchor in this mesh, since the neighborhood structure of the other four extraordinary vertices are similar to each another. The edges traversed in the "particle-shooting" process are shown bold.](image-url)
4.4 A False Start: The Greedy Algorithm

Given a set of anchors to begin our matching process, perhaps the most natural heuristic algorithm for solving the approximate topological matching problem is to start with a seed pair of matching quads, starting at some candidate anchors, and grow out matching meshes from this starting point.

The natural greedy algorithm based on this approach would be to grow out the matching set of quads in waves, adding as many quads as possible so as to satisfy the adjacency constraint for quads (e.g., by restricting our attention to quads that are adjacent across an edge or vertex). Unfortunately, this greedy approach suffers from a serious drawback, which is highlighted by a simple example.

Suppose we are given two similar meshes, $M_1$ and $M_2$, as, for example, shown in Figures 4 and 5, such that $M_2$ is an exact match for $M_1$ except that some small group of quads in $M_2$ is compressed into a set of edges.

![Figure 5: A candidate quad mesh, $M_2$. Note that $M_2$ is an exact match for the quad mesh $M_1$ from 4 except that three quads in the middle are compressed into individual edges in $M_2$.]

Figure 5: A candidate quad mesh, $M_2$. Note that $M_2$ is an exact match for the quad mesh $M_1$ from 4 except that three quads in the middle are compressed into individual edges in $M_2$.

Unfortunately, this greedy approach suffers from a cascade of failure, which is caused by the incorrectly-matched quads being “grown” ahead of the correctly-matching quads. The correctly-matching quads never have a chance to “catch up” to this wave, however, and it can continue cascading across the entire mesh. This unfortunate growth pattern is illustrated for the meshes of Figures 4 and 5 in Figure 6.

![Figure 6: An unfortunate cascading failure in $M_2$ caused by the greedy mesh growing algorithm applied to the meshes of Figures 4 and 5.]

Figure 6: An unfortunate cascading failure in $M_2$ caused by the greedy mesh growing algorithm applied to the meshes of Figures 4 and 5.

4.5 The Lazy-Greedy Algorithm

In this section, we discuss our lazy-greedy approach to solving the approximate topological matching problem for two quad meshes by using a wavefront “fire-propagation” method.

4.5.1 The Main Idea Behind our Matching Algorithm

The main idea behind this oxymoronic algorithm is to grow out waves of matching quads, as in the greedy algorithm given above as a false start, but to do so in a more relaxed way that helps to avoid the cascading failures that can arise from the straightforward greedy algorithm.

Let us assume that we have done an initial color labeling and have found a set of anchors to begin the matching process from in $M_1$ and $M_2$. The goal of the lazy-greedy algorithm is to incrementally build a mapping function, $\mu$, that matches quads in $M_1$ to quads in $M_2$. Initially, $\mu$ maps the two anchor quads in $M_1$ and $M_2$ to each other (defining an edge in the dual graph). As we proceed, we track of the set of quads where we can expand the matching function $\mu$ to more quads. Let $S$ be the set of currently matched quads, that is, each quad $q$ in $M_1$ for which we have determined a matching quad, $\mu(q)$. Since we start matching from a seed, at each iteration, $S$ forms a contiguous block of quads in $M_1$ with a corresponding set of matching quads, $\mu(S)$ in $M_2$. Now, let $S'$ be a subset of quads in $S$ on the boundary of the contiguous submesh $S$, that is, quads that have adjacent unmatched neighbors. When growing the match at each iteration, note that we only need to consider quads that are either vertex or edge adjacent to quads in $S'$ or $\mu(S')$, since the quads that are interior to the contiguous block have no adjacent unmatched quads.

4.5.2 Using a Compatibility Graph for Growing Matching Submeshes

Let $A$ be the set of unmatched quads in $M_1$ that are vertex or edge adjacent to quads in $S'$, and let $B$ be the set of unmatched quads in $M_2$ that are vertex or edge adjacent to quads in $\mu(S')$. For each quad $q$ in $A$, let $M(q)$ be the set of quads in $B$ that could match
with \( q \), that is, we include in \( M(q) \) each quad \( r \) in \( B \) such that \( r \) is adjacent to a quad \( \mu(t) \) with \( q \) being adjacent to \( t \) in the same way as \( r \) and \( \mu(t) \) (i.e., across a corresponding vertex or edge adjacency).

Let us create a compatibility graph \( L \) by defining, for each \( M(q) \), a vertex \( v_{t,q} \) for each quad in \( M(q) \). Note that same quad in \( B \) might be listed in different \( M(q) \) sets, in which case we create a different vertex in \( L \) for each copy of that quad in the different \( M(q) \) sets. We say that a vertex \( v_{t,q} \) is adjacent to vertex \( v_{s,r} \) in \( L \) if

- \( q \) and \( s \) are adjacent across an edge in \( M_1 \).
- The quads, \( r \) and \( t \) in \( M_2 \), corresponding respectively to \( v_{t,q} \) and \( v_{s,r} \) are compatible in \( M_2 \), meaning that if we were to extend \( \mu \) by mapping \( q \) to \( r \) and \( s \) to \( t \), then the submesh in \( M_1 \) consisting of \( q \) and \( s \) and all their adjacent quads in \( S' \) would be consistent (in the topological sense) with the submesh in \( M_2 \) consisting of \( r \) and \( t \) and all their adjacent quads in \( \mu(S') \).

Note that the graph \( L \) consists of paths, isolated vertices, or a cycle, i.e., vertices in \( L \) have degree at most 2.

### 4.5.3 The Lazy-Greedy Heuristic

The lazy-greedy heuristic is to extend \( \mu \) in each iteration by adding the matches defined by a largest connected component (i.e., path or cycle) in \( L \). Note that this is a greedy algorithm in the sense that it is augmenting our match using an optimization criterion that maximizes an objective function. But it is also a lazy algorithm in that it postpones performing a lot of potentially valid matches.

The benefit of the lazy-greedy approach is that it allows our matching process to match the quads around a small mismatching region even when the mismatch is caused by a compression of quads into individual edges. Such a compression causes the greedy algorithm to immediately march through these bad regions, whereas the lazy-greedy algorithm will only venture into such regions as a last resort. We illustrate the difference this approach makes for the meshes \( M_1 \) and \( M_2 \) of Figures 4 and 5 in Figure 8.

![Figure 8: An isolation of a small mismatching region in \( M_2 \), showing how the lazy-greedy algorithm avoids the unfortunate cascading failure caused by the greedy mesh growing algorithm applied to the meshes of Figures 4 and 5.](image)

<table>
<thead>
<tr>
<th>Model</th>
<th>Original mesh</th>
<th>Skeleton graph</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vertices</td>
<td>edges</td>
<td>vertices</td>
</tr>
<tr>
<td>bear</td>
<td>1070</td>
<td>2110</td>
<td>202</td>
</tr>
<tr>
<td>chic</td>
<td>932</td>
<td>1827</td>
<td>32</td>
</tr>
<tr>
<td>shirt</td>
<td>3099</td>
<td>6134</td>
<td>518</td>
</tr>
<tr>
<td>director</td>
<td>9958</td>
<td>19889</td>
<td>2510</td>
</tr>
<tr>
<td>tommy</td>
<td>10281</td>
<td>20530</td>
<td>2496</td>
</tr>
<tr>
<td>body</td>
<td>13238</td>
<td>26457</td>
<td>3176</td>
</tr>
<tr>
<td>girl</td>
<td>6976</td>
<td>13903</td>
<td>2340</td>
</tr>
</tbody>
</table>

Table 1: Sizes of skeleton graphs for various quad mesh models.

Note that the reduction percentages average around 75%. The main point of this compression is not for storage savings, however, although it could be used for this purpose, since the regions bounded by edges of the skeleton graph are all structured meshes. Instead, we use the skeleton graph to drive our approximate matching process.
Another claim of our method is the following:

- The lazy-greedy algorithm runs fast enough for interactive modeling purposes.

The running times for our matching process, applied to the same characters as used for skeleton graph computations are shown in Table 2. The second meshes in these cases correspond to similar models (e.g., two bears), alternate poses, and across symmetries (e.g., in a shirt).

<table>
<thead>
<tr>
<th>Model</th>
<th>Mesh size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vertices</td>
<td>edges</td>
</tr>
<tr>
<td>bear</td>
<td>1070</td>
<td>2110</td>
</tr>
<tr>
<td>chic</td>
<td>932</td>
<td>1827</td>
</tr>
<tr>
<td>shirt</td>
<td>3099</td>
<td>6134</td>
</tr>
<tr>
<td>director</td>
<td>9958</td>
<td>19889</td>
</tr>
<tr>
<td>tommy</td>
<td>10281</td>
<td>20530</td>
</tr>
<tr>
<td>body</td>
<td>13238</td>
<td>26457</td>
</tr>
<tr>
<td>girl</td>
<td>6976</td>
<td>13903</td>
</tr>
</tbody>
</table>

Table 2: Running times for approximate topological matching of various quad mesh models.

Note that all the running times for approximate matching are at a second or less. This is certainly sufficient for interactive modeling purposes. Moreover, it significantly speeds up what used to be a semi-automated process that involved human input of a pair of edges to use as anchors.

Finally, we also make the following claim for our algorithm:

- The lazy-greedy algorithm finds good approximate matches.

Naturally, since we are not aware of any other approximate topological matching algorithms, testing this claim against other algorithms is not possible. Nevertheless, we can prove that our algorithm is successful in many cases. Suppose, for example, that there is a mapping between two simple, connected, well-formed quad meshes $M_1$ and $M_2$ such that each connected mismatched region has boundary complexity of at most $\delta$. Suppose further that the medial axis of the matched region for $M_1$ and $M_2$ is connected and has a spanning tree $T$ such that each edge of $T$ has a cross-sectional width of at least $\epsilon > 2\delta$. Then our algorithm will succeed in finding a match at least as good as this match.

In addition, a subjective evaluation of the matches our algorithm produced on the Disney character database demonstrated that our algorithm empirically found good matches. For example, we show the quality of the matches produced by our algorithm on the bears and girl models in Figures 9 and 10. In addition, an animation of our algorithm is available for download at the following location:

www.ics.uci.edu/~eppstein/projects/firefront.mov

6 Conclusion and Future Directions

In this paper, we study the approximate topological matching problem for quad meshes, showing this problem is NP-hard in general. Nevertheless, we provide an efficient heuristic algorithm that works well in practice and provably well for a large class of models that is common in practice.

Directions for future work include the following:

- Our methods are directed at connected models. How hard is it to extend the lazy-greedy algorithm to handle disconnected models?

- Our methods find a single connected region of matching quads. Is there a way to generalize the lazy-greedy approach to handle disconnected matching regions in a way that maintains distance and intuitive correspondence as much as possible?
Acknowledgments

This research was done, in part, while the first two authors were consultants for Walt Disney Animation Studios. Likewise, this research was done, in part, while the third author was a summer intern for Walt Disney Animation Studios.

References


