Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

Curvature Aware Fundamental Cycles

P. Diaz-Gutierrez*[1] and D. Eppstein[1] and M. Gopi[1]

[1]University of California, Irvine

Abstract

We present a graph algorithm to find fundamental cycles aligned with the principal curvature directions of a surface. Specifically, we use the tree-cotree decomposition of graphs embedded in manifolds, guided with edge weights, in order to produce these cycles. Our algorithm is very quick compared to existing methods, with a worst case running time of $O(n \log n + gn)$ where n is the number of faces and g is the surface genus. Further, its flexibility to accommodate different weighting functions and to handle boundaries may be used to produce cycles suitable for a variety of applications and models.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

Figure 1. Fundamental cycles generated by our flexible control of cycle computation on a triangle mesh with 50,000 faces. The six cycles computed in this genus 3 model were found using a simple breadth-first construction of the spanning tree to guide our algorithm.

1. Introduction

There are many applications of algorithms for understanding the topology of geometric models, and they continue to grow as the field develops. Examples include surface parameterization, faithful surface reconstruction, morphing across arbitrary shapes and even computational chemistry. Among the most useful topological shape descriptors are the fundamental cycles of a surface M – classes of closed paths on M that cannot be continuously retracted into a point (they are not trivial in M). These curves provide useful information that can be used to determine the compatibility of the surface shape with that of other surfaces; additionally, small cycles may indicate regions of incorrect surface reconstruction, or they may represent insignificant details whose elimination will allow for higher-quality simplification of the model.

A simple flooding algorithm on the faces of a mesh representation of the surface can be used to find fundamental cycles, but the cycles found in this way are not well shaped and not very useful for many applications. On the other hand, algorithms that find minimum-length fundamental cycles exist but they are more expensive [CM07], exhibiting a computational complexity of $O(g^{3/2}V^{3/2} + g^{5/2}V^{1/2})$, or the recent [ZJLG09] which computes a representative for each homotopy class. Not much earlier, an elegant algo-

* Currently working for Intel Corporation

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

rithm [DLSCS08] was presented to produce geometrically aware fundamental cycles. This method can also distinguish the curves between handles and tunnels, as it tetrahedralizes the space in which the surface is embedded. However the tetrahedralization step of this method may itself be expensive, and this method requires the input model to be watertight. The method we present does not require information about the space in which the surface is embedded, and therefore is unable to determine whether the cycles produced are tunnels or handles.

Our intent is to develop algorithms for fundamental cycles that are highly efficient, that are robust in the face of minor defects in the model, and that allow the user the flexibility to specify the desired fundamental curves for any particular application. We provide a simple mechanism to indicate the priority of the edges of the shape that are more desirable to the user. An efficient and robust graph algorithm finds and evaluates a gamut of solutions and selects those that meet the given criteria and constraints. Notably, our method can also handle manifolds with boundaries without any preprocessing. We demonstrate the usability of our approach by efficiently producing fundamental cycles that follow the general directions of minimum and maximum curvature.

1.1. Main Contributions

The following are the main contributions of this paper:

- We present a method to guide a tree-cotree decomposition to produce fundamental cycles on manifolds with or without boundaries following user recommendations.
- We propose an efficient method to evaluate all the possible fundamental cycles allowed by the algorithm, and to efficiently pick those that best fit the recommendations.
- We analyze the properties of the possible cycles produced by the previous algorithm, in order to quickly determine whether they are contractible, separating or fundamental.
- We demonstrate our method by using an edge weighting scheme to produce curvature aware fundamental cycles, naturally producing curves that adapt to the foldings of the processed shape.

As part of our algorithm, we require a listing of the bridges of a graph, a problem that can be solved by the classic biconnected graph component algorithm [Tar72]. We describe an easy-to-implement version of this algorithm that lists all bridges in linear time.

This paper is organized as follows. We begin with a brief review of the relevant literature. Next we introduce the tree-cotree decomposition, which forms the basis for our contributions in this chapter, along with the outline of an algorithm to quickly evaluate the possible fundamental cycles enabled by a given decomposition. We continue by describing the techniques that we use to guide the decomposition and hence to compute the fundamental cycles of the input shape. Finally, we discuss relevant implementation details and show the results of our implementation.

2. Previous Work

The potential for topology understanding algorithms is vast and growing, not only in computer graphics and computational geometry, but in unrelated areas like computational chemistry [LEF*98]. A few of the applications and algorithms that analyze and process topology include surface parameterization [GY03], generating topologically correct isosurfaces from scalar data [Nat92, SV03], simplification of excess genus [WHDS04], representation via *geometry images* [GGH02], as well as topology independent morphing and cross-parameterization [LYC*06]. Fundamental to all these applications and algorithms is the computation of different topological descriptors. Here we review a few of such algorithms with a focus on fundamental cycles. We also review the background for the tree-cotree decomposition, the algorithm on which we base our contributions.

Topological descriptors: One important milestone in computational topology was presented by [DE93], for the computation the Betti numbers of a geometric 3-dimensional complex, following a more general study [DC91] which tackles simplicial complexes. Soon, the formalization of the field [DEG99] encouraged researchers to use similar incremental algorithms for their computation and revisit concepts from graph theory related to topologically embedded graphs. The recent surge of interest in Reeb graphs [PSBM07, PSF08] since they were first introduced to computer graphics [SKK91] represents a good example. Originally defined in 1946, Reeb graphs [Ree46] combine fundamental cycles with other curves derived from Morse function analysis, offering a succinct representation of the geometric and topological structure of a shape. The fundamental cycles of the shape, which are the main topological element of our interest, can be extracted [CMEH*03] from the Reeb graph of a shape. However, although some methods exist to produce a less complex Reeb graph [NGH04] by modifying the scalar function that generates the graph, by being tied to the singular values of a function defined on the surface, they lack flexibility in the design of these curves. Recent work [DLSCS08] uses an incremental algorithm to compute geometry aware handles and tunnels. For the classification of the fundamental cycles into handles and tunnels, the object and the embedded space have to be tetrahedralized and analyzed.

From the point of view of graph theory, relevant results have been known for decades, although variations of the problem of computing optimal cycles under different measures are still under active development. Some methods aim at producing the generating cycles [Epp03] in a purely combinatorial way. The problem of finding the shortest set of cuts to split a closed manifold into a set of topological disks is

known to be NP-hard [EHP02], and so is the related problem of finding the shortest splitting cycle on a combinatorial surface [CCdVE*08]. On the other hand, the complementary problem of finding the shortest non-separating and non-contractible cycles is expensive but tractable [CM07]. For these reasons, it is common to start with a rough approximation and then converge to a local minimum in some geometric measure *a posteriori*, typically geodesic cycle length [YJG07,CdVL07]. A similar approach is to turn the problem into finding the shortest set of fundamental cycles passing through one [YJG07] or several [CC07] given points in the surface.

Tree-cotree decomposition: In a planar graph, the complementary set of edges to any spanning tree forms a spanning cotree (spanning tree of the dual graph) [KR87]; this fact was used, e.g., by Von Staudt in 1887 to prove Euler's formula $v-e+f=2$ [Som58]. Several algorithmic problems on planar graphs benefit from partitioning the edges of the graph into a spanning tree and a spanning cotree; these include dynamic maintenance of minimum spanning trees and connectivity information [EIT*90], point location data structures [GT91], and geometric group theory [GR05, RT06].

For a graph embedded on a surface of higher genus than the plane, Eppstein defined and proved the existence of *tree-cotree decompositions* [Epp03] in which the edges of the graph are partitioned into three sets: a spanning tree of the primal graph, a spanning tree of the dual graph, and a small number of leftover edges. When applied to graphs embedded in orientable surfaces of genus $g>0$, the decomposition leaves exactly $2g$ edges that do not belong to either the spanning tree or the spanning cotree. These edges are the key to producing the fundamental cycles of the embedding surface. Tree-cotree decompositions can be used to efficiently compute and maintain the generators of dynamic graphs [Epp03] and to find short generating sets with a fixed basepoint [EW05]. As we will see below, fundamental cycles that follow different design criteria can be computed by appropriately guiding the construction of the spanning tree and cotree.

3. Computing Fundamental Cycles

In this section we introduce a method for computing the fundamental cycles of a manifold with or without boundaries. We also describe techniques that allow the user to indicate the desired direction that the cycles ought to follow. Finally, we present algorithms to efficiently select the shortest among all the possible cycles satisfying the user requirements.

3.1. The Tree-Cotree Decomposition

A graph formed by all the vertices and edges of the polygonal mesh is called the **primal graph**. Its **dual graph** has one node for each mesh face, and an edge between two nodes if their associated mesh faces share a mesh edge. We call a spanning tree of the primal graph just **spanning tree** for simplicity. A spanning tree of the dual mesh graph is called a **spanning cotree**.

Definition 1. ***Tree-cotree decomposition:*** *A tree-cotree decomposition of a mesh is a partition of the set E of edges of a mesh into three sets $(T, C, E \setminus (T \cup C))$, where T is the set of edges in the spanning tree and C is the set of mesh edges corresponding to the edges of the spanning cotree.*

The tree-cotree decomposition can be computed directly from the definition in linear time on the size of the input mesh. First a spanning tree T is constructed. Every spanning tree is part of a tree-cotree decomposition [Epp03], so the remaining part of the decomposition may be found by computing a spanning tree C for the subgraph of the dual graph of the mesh formed by the edges in $E \setminus T$. Since the tree-cotree decomposition is a partition of the mesh edges, $T \cap C = \emptyset$.

Let us briefly analyze the properties of this decomposition. Let V, E and F be the number of vertices, edges and faces of the input mesh, respectively. A spanning tree has exactly $V-1$ edges, while a spanning cotree contains $F-1$ edges. Applying Euler's characteristic for a genus g surface ($V-E+F=2-2g$), since the mesh edges corresponding to spanning tree and cotree are disjoint, the number of leftover mesh edges in the tree-cotree decomposition is $2g$. Each of these leftover edges, when added to the spanning tree, creates a cycle – a non-separating fundamental cycle. In Figure 2 we can see a tree-cotree decomposition and the resulting fundamental cycles.

If the input is a manifold with boundaries, holes are handled transparently by this algorithm. The above calculations remain correct by considering every boundary loop to enclose a hypothetical face. The geometry of this hypothetical face does not need to be determined. This is a significant distinction from other algorithms that require a well defined inside and outside partition of the embedding space for correctness.

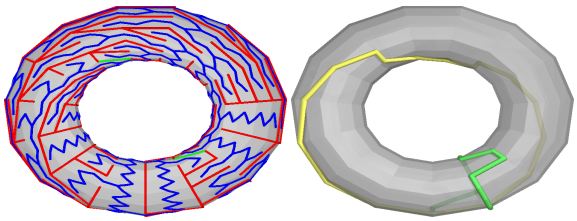

Figure 2. **Left:** Tree-cotree decomposition of a toroidal triangle mesh, showing the spanning tree (red) and the spanning cotree (blue). When added to the spanning tree, each leftover edge (green) induces a fundamental cycle. **Right:** Fundamental cycles resulting from the previous decomposition.

The key new idea behind our method is to assign weights to mesh edges to guide the spanning tree and spanning cotree construction. Prominent user criteria are expressed as edge

weights in order to guide the method to the formation of cycles that are close to the desired curves. In our application described in Section 5, we use the deviation of the edges from the minimum or maximum curvature direction on the mesh as edge weights. The reason for this choice is that the minimum and maximum curvature directions concisely describe the folding structure of the shape, and are thus useful for multiple applications where surface cutting needs to adapt to these features. While those criteria that can be expressed as edge weights can be taken care of this way, applying certain global constraints like shortest loops requires evaluating many cycles to choose the best one. We present an efficient graph algorithm that uses *lowest common ancestor* queries to measure the length of all possible fundamental curves within this smaller set of cycles that satisfy the user criteria. In the process, we also extend the classic algorithm by [Tar72] to find and eliminate those edges in the dual graph that do not generate any fundamental cycle in the primal. As a result we create the best $2g$ number of fundamental cycles of the given mesh for a given edge weighting scheme. This method is explained in detail in Section 3.2.

3.2. Candidate Connection Detection

Given a set of edge weights that express the preferred direction of the fundamental curves, we compute a minimum spanning tree T on the mesh graph and a maximum spanning tree C on the dual graph of the mesh; these two trees are known to be disjoint [Epp03]. The complexity of this part of the algorithm, using simple and easy to implement minimum spanning tree algorithms such as Kruskal's algorithm, is $O(n \log n)$ where n is the number of faces (or edges) in the input triangular mesh; even linear time algorithms exist for cases with low genus. Faster time bounds may be achieved by using more complex minimum spanning tree algorithms [KKT95]. The edges left over from both the spanning tree and cotree, when added to the spanning tree will form the desirable fundamental cycles. Each cycle is formed by one of these leftover edges together with the path in the primal spanning tree connecting the edge's two endpoints.

The above algorithm, although guaranteed to produce topologically correct cycles, can produce fundamental cycles of vastly different geometric shapes with a small change in the location of the chosen leftover edges (see Figure 3). In other words, the edge weights alone, though they help steer the direction of the fundamental cycles locally, cannot optimize a global criterion to produce geometrically well shaped curves. To produce cycles of better shapes, we modify the decomposition, keeping the minimum spanning tree T but replacing C by a different cotree. In this, we diverge from previous works like [Epp03, EW05], which explicitly compute the maximum spanning cotree. In the process, we evaluate all the fundamental cycles that could be created as part of a tree-cotree decomposition involving T. Every edge in $E \setminus T$ will create a cycle in the spanning tree, but only some of them will create fundamental cycles; our goal is to quickly find the mesh edges (also called *candidate connections*) that create fundamental cycles, evaluate those cycles, and choose the best cycle.

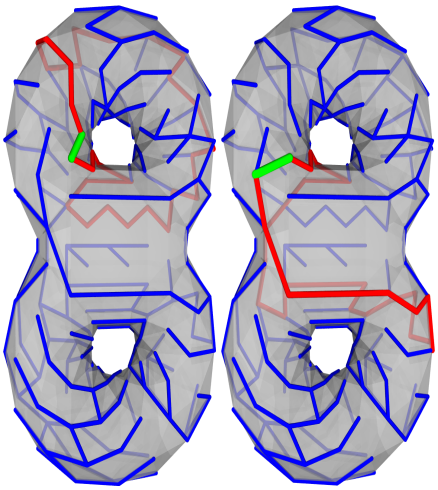

Figure 3. A given spanning tree (blue) in the tree-cotree decomposition can form very different fundamental cycles (red) with the choice of even adjacent candidate connections (green).

Definition 2. ***Cograph:*** *A subgraph of the dual graph that does not include the dual edges corresponding to the edges of the spanning tree.*

In other words, the cograph is the spanning cotree along with the edges corresponding the the leftover edges in the tree-cotree decomposition.

Definition 3. ***Candidate connection:*** *Given a mesh M and a spanning tree T, a mesh edge e is called a candidate connection if $T \cup \{e\}$ produces a fundamental cycle in M.*

Specifically, any mesh edge whose corresponding dual edge is *not* a *bridge edge* of the cograph is a candidate connection and would induce a non-separating fundamental cycle in the primal graph. Bridge edges of graph G are those which, if removed, would split G into disjoint components; a bridge edge must belong to any spanning tree of G, so the bridge edges of the dual graph must belong to the cotree C and cannot be used to generate fundamental cycles in our tree-cotree decomposition. Bridge edges can be of two types, as illustrated in Figure 4: *Dangling bridges* are the edges which are removed if we iteratively "shave" all the degree-1 vertices. All other bridge edges are called *internal bridges*. This classification has interesting implications. Mesh edges corresponding to dangling edges in the co-graph induce contractible cycles in the primal graph, while those of internal bridges induce non-contractible, but separating cycles. Neither of them are desirable to use as fundamental cycles. The following algorithm identifies all the bridge edges and removes them. Once these edges have been removed, the remaining edges are our desired candidate connections.

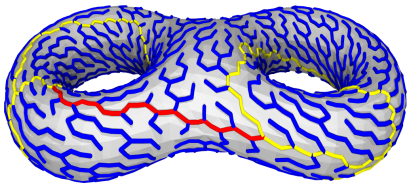

Figure 4. Dangling bridges (blue) and internal bridges (red) on the cograph of a tree-cotree decomposition. Unlike the candidate connections (yellow), bridge edges do not induce fundamental cycles. Instead, they induce contractible and separating cycles, respectively.

In order to find the bridge edges, we extend a classic biconnected graph components algorithm [Tar72] which can also be used to find the graph articulation points (a concept analogous to bridge edges, but with vertices). We begin by computing a graph G' derived from the original cograph G. Each edge in G produces a node in G', and two nodes in G' share an edge if their corresponding edges in G share a common node. In this derived graph G', the articulation points correspond to bridge edges in G. This algorithm for the identification of all the bridge edges in the cograph has $O(n)$ time complexity with linear memory requirement.

3.3. Candidate Connection Evaluation

The candidate connections in the cograph as found above induce fundamental cycles in the primal. Our measure of evaluating each of these fundamental cycles is its hop-length. We use the the *lowest common ancestor* (LCA) algorithm to measure the length of all fundamental cycles. Let a candidate connection e connect two nodes e_a and e_b of the spanning tree T. There is a unique path $p(e_a, e_b)$ in T from e_a to e_b. If we pick an arbitrary vertex r to be the root of T, then the LCA of two nodes, denoted by $lca(a,b)$ is the root of the smallest subtree that contains both e_a and e_b. The path $p(e_a, e_b)$ goes up from e_a to $lca(e_a, e_b)$ and back down to e_b. The fundamental cycle induced by edge e is $p(e_a, e_b) \cup \{e\}$, and its total length, measured as the number of edges it contains, is given by $1 + depth(e_a) + depth(e_b) - 2depth(lca(e_a, e_b))$. Each query for LCA of two nodes can be answered in a constant time using an appropriate data structure [BFC00, GT83, HT84, SV88] that is constructed in linear time in the size of the tree, which in our case is $O(n)$. Furthermore, the length of the fundamental cycles corresponding to each candidate connection can be precomputed and stored, since this value is fixed.

This method evaluates the length of a graph cycle as the number of edges it contains. If the length of the edges is uniform, this is a good approximation of the total geodesic length. Otherwise, we can incorporate geodesic distance to the LCA algorithm by replacing the node depth in the computations of cycle length by the distance from the root of the tree, at no additional cost in asymptotic time complexity.

Finally, cycles can be further shortened using a simple local optimization. If a cycle goes through two edges of a triangle, it can be modified to pass through the third edge of the triangle. Wider searches can be applied if desired, always looking for a shorter way to connect two near points along the curve. This method can be modified to locally re-route cycles along lower cost paths, as per by the user given weights, instead of just shorter paths.

3.4. Candidate Connection Selection

All candidate connections, when added to the spanning tree, produce fundamental cycles. But only $2g$ of them belong to distinct homotopy classes. The rest are topologically equivalent to these $2g$ cycles, or to combinations of these cycles. The goal is to find these best $2g$ candidate connections and hence the best $2g$ fundamental cycles.

We perform this selection using a greedy algorithm, as follows. We choose the best candidate connection as evaluated in Section 3.3 as our first connection and remove it from the cograph. This removal might cause many others of the candidate connections to become bridge edges. These newly formed bridge edges are exactly the candidate connections that would induce fundamental cycles of the same homotopy class as the chosen candidate connection. We then run the modified Tarjan's bi-connected graph component algorithm (Section 3.2) to remove these new bridge edges too. We repeat the above process of selection of candidate connection and removal of bridge edges, exactly $2g$ times, at which point there will be no more candidate connections to be considered. The result is a tree-cotree decomposition in which we have selected the tree T first, the $2g$ generating edges $E \setminus (T \cup C)$ second, and in which the cotree C consists of any remaining edges not selected to generate cycles, in contrast to the previous method of finding T and C first. Performing the choices in this order gains us greater control over the shapes of the cycles we find. The time complexity of this entire method is $O(gn)$ since we have to run the linear-time biconnected graph component algorithm $2g$ times. When taken as a whole, our algorithm produces non-optimal results in terms of cycle length, unlike methods like [EW05]. On the other hand, our method has an easy implementation and performs asymptotically faster. Furthermore, as shown in Section 4, it produces cycles that in practical situations can be paired as complementary, in a way similar to that of [DLSCS08].

4. Extensions

Our main algorithm, presented in earlier sections, is based on the tree-cotree decomposition, and inherits all of its properties. In this section we present extensions to our algorithm that, although they are not theoretically proven to work all the time, we could not find a practical case when they failed.

4.1. Complementary Cycles

The above method produces the required $2g$ fundamental cycles for a specific edge-weighting scheme. We can process these cycles to pair 'complementary' cycles, under certain situations, for their further use in merging of two sets of fundamental cycles produced under different edge-weighting schemes.

Many simple models may be formed by attaching disjoint handles and tunnels to a sphere, such that the pair of fundamental cycles describing each feature are disjoint from the other cycles describing the other features, as shown in Figure 5; such a family of cycles is known as a *canonical schema* [LPVV01], and it exists for any closed 2-manifold.

When the cycles we find form a canonical schema, we may determine which pairs of cycles are mutually complementary (informally, they are the tunnel and handle of the same topological feature), by counting the number of times a pair of fundamental cycles cross each other: if they intersect an odd number of times, then they *can be* complementary [DLS07] (see Figures 1 and 5). The simple case, a canonical schema, can be detected as the situation in which each fundamental cycle intersects an odd number of times with exactly one other fundamental cycle. It is true in general that not all sets of fundamental cycles can be paired up by this odd cross-count measure. But our algorithm, directly derived from the tree-cotree decomposition, only generates cycles which are indeed paired in such way, and hence we have not found practical cases where the algorithm fails.

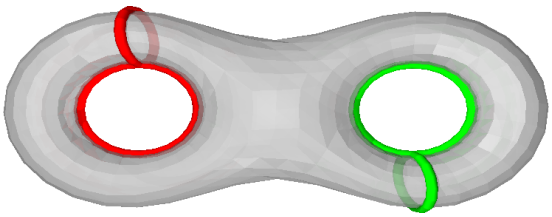

Figure 5. Fundamental cycles paired by a relationship of complementarity. Cycles colored the same way are complementary handles and tunnels. Without considering the embedding space, it is not possible to tell which one is which.

To test quickly whether two cycles intersect an odd number of times, and to properly handle the situation in which two cycles overlap in a set of edges rather than crossing at a finite vertex set, we take advantage of the fact that our cycles all have the form of a single candidate edge added to a path in a common spanning tree T. To do so, we form the preorder and postorder vertex sequences of T (in both cases traversing the children of each vertex in T in clockwise order according to the local orientation of the model). A cycle formed by the candidate edge (u,v) and a second cycle formed by the candidate edge (w,x) necessarily cross an odd number of times if and only if these four vertices occur in the same sorted order in both preorder and postorder, and this sorted order interleaves the endpoints of the two edges; for instance, if the vertices appear in the order u, w, v, x in both preorder and postorder. If the sorted order of the vertices is different in preorder and postorder, then there is an ancestor-descendant relationship among two of the vertices, and the cycles may be perturbed so that they cross an even number of times. And if the sorted order of the vertices is the same in preorder and postorder, but not interleaved, then the cycles necessarily cross an even number of times.

Thus, after $O(n)$ preprocessing (computing the preorder and postorder numberings) we may detect whether any two cycles intersect an odd number of times in constant time. We may then test all pairs of cycles, and detect the situation in which each cycle intersects an odd number of times with exactly one other cycle, in a total of $O(g^2)$ time, smaller than the $O(gn)$ time taken by our algorithm for finding the set of fundamental cycles. This method successfully pairs cycles of the canonical schema into complementary cycles since each cycle has a unique pair with which it has an odd intersection.

For more complicated models and sets of fundamental cycles that cannot be uniquely matched in this way, we may nevertheless partition the cycles into complementary pairs, as follows. Form a graph (not necessarily bipartite) in which the vertices represent fundamental cycles and the edges represent pairs of cycles that intersect each other an odd number of times, and match cycles into complementary pairs by finding a perfect matching in this graph. Using the same preorder and postorder numbering technique described above to test the number of crossings of each pair of cycles, we may construct the graph in time $O(g^2+n)$. Finding a perfect matching in the resulting unweighted non-bipartite graph may be performed in time $O(g^{5/2})$ [MV80] (or in time $O(g^3)$ using easier to implement methods such as Edmonds' blossom-contraction algorithm [Edm65, Gal86]). Therefore, including the time to construct the tree-cotree decomposition, this algorithm takes a total time of $O(gn+n\log n+g^{5/2})$. Unless the genus is very large ($g > n^{2/3}$ for the $O(g^{5/2})$-time matching algorithm, or $g > n^{1/2}$ for the cubic-time algorithm), the time for the matching step is dominated by the time for our tree-cotree decomposition algorithm.

4.2. Merging Sets of Fundamental Cycles

In some applications, the two cycles in each pair of fundamental cycles may be required to follow two contradictory goals. In order to satisfy them, the algorithm for generating sets of fundamental cycles can be run for two different edge weighting schemes (for example, once using a weighting scheme that forms a tree T with edges in the direction of minimum local curvature, and a second time using a different weighting scheme that forms a tree T with edges in the perpendicular direction of maximum local curvature). In each run we get $2g$ fundamental cycles. We would like to pick $2g$ out of the total of $4g$ cycles with the following goal:

In the chosen $2g$ cycles there are exactly g complementary pairs and in each of the complementary pairs, one cycle follows one weighting scheme (say, minimum curvature direction) while the other cycle follows the other scheme (say, maximum curvature). A simple example of this operation is illustrated in Figure 6.

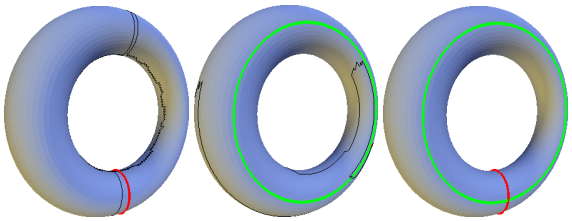

Figure 6. Cycles generated along maximum (left) and minimum (middle) curvature directions, respectively. Among each set of cycles, those highlighed in color are picked by the cycle merging algorithm (right).

To see why this problem is, in general, difficult, consider the simplest case, the torus with $g = 1$. Any cycle in the torus may be represented by a pair of integers (a,b), where a describes the number of times the cycle wraps around the handle of the torus and b describes the number of times the cycle wraps around the tunnel. This representation describes an equivalence between the fundamental group of the torus and the additive group of two-dimensional integer vectors. Two vectors (a,b) and (c,d) generate the entire two-dimensional integer lattice if and only if the determinant $ad - bc$ is ± 1; geometrically, this corresponds to the two corresponding cycles having *signed crossing number* ± 1, where the signed crossing number of one oriented cycle with respect to the other is determined by adding $+1$ for each left-to-right crossing and -1 for each right-to-left crossings. Thus, for instance, if one pair of fundamental cycles is represented by the numbers $(1,0)$ (a handle) and $(0,1)$ (a tunnel), while the second pair of fundamental cycles is represented by the numbers $(2,3)$ (a curve that spirals around the torus wrapping twice around the handle and three times around the tunnel) and $(3,4)$, then there can be no way of choosing one cycle from the first pair and one cycle from the second pair in order to form a combined basis, because each of the four determinants of one vector from $\{(1,0),(0,1)\}$ and one vector from $\{(2,3),(3,4)\}$ is too large.

In cases when complementary pairs do not exist across two different sets of cycles as explained above, either one of the two sets of $2g$ cycles is still a valid solution. In common cases, given two sets of $2g$ cycles, we may group them into complementary pairs (as in the previous section), and form a bipartite multigraph G^P with one node per complementary pair. We include an edge between two nodes of G^P when there is a way of choosing a cycle from each of the two corresponding complementary pairs in such a way that the two chosen cycles have signed crossing number ± 1. In this case, we use the sum of the lengths of the two cycles as the weight of the edge in G^P. A minimum weight perfect matching in G^P will give us a collection of g complementary pairs of cycles from the $4g$ cycles of the union of the two sets. Computing each crossing number takes $O(n)$ time, and the graph G^P has $O(g)$ vertices and $O(g^2)$ edges, so the total time for this matching algorithm is $O(g^2n+g^3) = O(g^2n)$. We get a complete set of $2g$ fundamental cycles when each cycle chosen by this algorithm has zero signed crossing number with each other cycle except the one to which it is matched. Thus we can merge the cycles found by two different edge-weighting schemes, into a single set of complete fundamental cycles.

5. Curvature Aware Fundamental Cycles

The entire algorithm explained in Section 3 requires a weighted mesh as input. In this section we explain our method for assigning edge weights to find curvature aware fundamental cycles. First we compute the principal directions at every vertex on the mesh, and use these directions to assign edge weights.

We estimate the principal direction vector field via principal component analysis of quadric matrix constructed at every mesh vertex [GH97]. More sophisticated methods exist, but we pick this one for its simplicity. Given a polygon normal $N_p = (a,b,c)^T$, its associated 3×3 quadric matrix is defined as the product $Q_p = N_p N_p^T$. A quadric matrix at a vertex v is given by the sum of the quadric matrices of all its incident polygons $Q^v = \sum_p Q_p$. The first two eigenvectors of Q^v indicate the directions of minimum and maximum surface curvature, respectively. Then we smooth this vector field using Laplacian filtering to minimize the effect of the inherent unstability of the principal curvature on a surface. The constraint that minimum and maximum curvature vector fields are mutually orthogonal is imposed to prevent inconsistent vector drift across the two fields. We then weight each edge by the average angle the edge makes with one of the vector fields (say, minimum) at its incident vertices. For our purposes, the sign of the vector field is irrelevant in angle calculation. This weighted mesh is used in our algorithm to find the fundamental cycles. A second set of fundamental cycles is computed by assigning edge weights based on the maximum curvature direction vector field. Cycles can be chosen from either set to produce a final set of fundamental cycles as needed by the application at hand.

Figure 7 shows the two vector fields on the mesh, the edge weighting scheme for minimum curvature direction, and the fundamental cycles chosen by our algorithm.

One of the most important complications of fair vector field design on surfaces is dealing with twirls and other singular points in the field. Their potential effect in our algorithm would be that of randomly redirecting fundamental cycles that enter the twirls. To prevent this, we first detect these singular vector field regions by measuring the vector field distortion with respect to adjacent mesh vertices, and thresh-

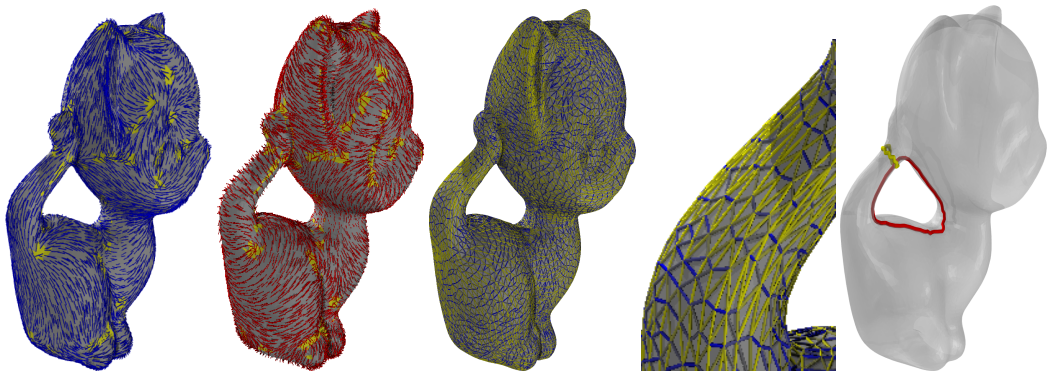

Figure 7. Smoothed minimum (blue) and maximum (red) curvature direction vector fields. The middle image shows the edge weights with respect to the minimum curvature direction. Higher priority is shown in yellow and lower priority in blue. The detail of the tail region shows that the edge weights are well aligned with the intended curvature direction. Finally, the figure on the right show a yellow cycle coming from a maximum curvature optimization, whereas the red cyle is the result of a minimum curvature weighing. The mesh contains 22,000 faces.

olding them. High turbulence regions are given the highest priority to be included in any spanning tree. This effectively makes the twirl regions "hubs" of the spanning tree, thus removing their influence in the general direction of the fundamental cycles. In Figure 7, these hub regions are shown in yellow.

6. Results

We tested our algorithm in several models with wide variety of geometric and topologic complexity. The results are shown in Figures 7 through 11. An experimental implementation running on a low end laptop (Intel Centrino 1.66 GHz, 512MB RAM) took less than two minutes to finish processing the largest model shown, with 480,000 faces. That includes all the preprocessing steps such as model loading, vector field estimation and smoothing. This is a substantial improvement over the state of the art, with algorithms often requiring several hours to finish.

Edge weights were the angle between the direction of the edge with the value of an underlying tangent vector field. Though there is no restriction on the type of vector field used for this purpose, we chose a smooth curvature frame in order to produce pleasant curves. On one hand, we have a spanning tree that connects edges along a pre-determined general direction, and on the other hand we pick the shortest cycle that can be generated using this spanning tree. These apparently conflicting goals produce results that wrap naturally around the folds of the shape. In summary, the computational aspects of our algorithm are entirely flexible. They are flexible in terms of the edge weighting scheme and also the way we build the spanning tree in an unweighted case. For example, a simple breadth-first construction of the tree has produced the results shown in Figure 1. All these aspects of weight computation are transparently replaceable components of our algorithm.

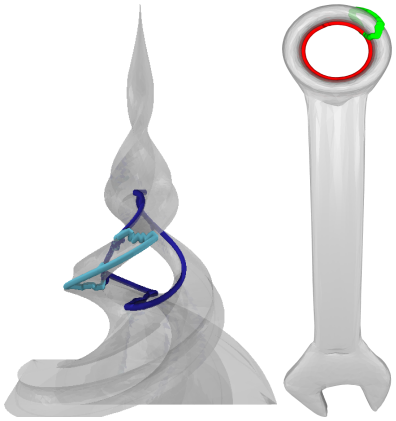

Figure 8. Left: Fundamental cycles on a screw model (10,000 faces), following the direction of minimum curvature. Notice how the curves snap to the edges, where the orientation of the curvature directions is more sharply defined. **Right:** Cycles on the wrench model (8,000 faces) using just one weighting scheme followed by simple tightening of the cycles (a curve that passes through two edges of a triangle is made to pass through only the third edge).

7. Conclusion

We have introduced the use of the tree-cotree decomposition for the guided computation of the fundamental cycles of a manifold with or without boundaries. This method can be driven to produce cycles that satisfy any user requirements, as long as they can be expressed as a set of edge weights in the graph embedded in the surface. We demonstrate our method by designing a weighting scheme that produces curves following the principal curvature directions. Finally, our method is very easy to implement and it can be extended using edge weighting schemes suitable for other applications.

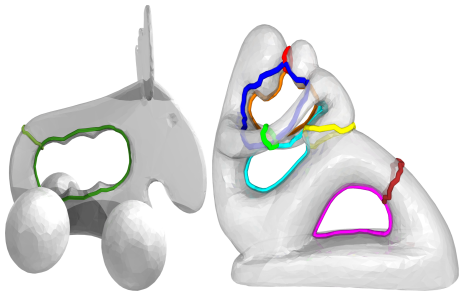

Figure 9. The elk (left, 10,000 faces) and fertility (right, 480,000 faces) models feature cycles from two different executions of our algorithm: once for edge weights favoring minimum curvature direction, and once for the maximum curvature direction.

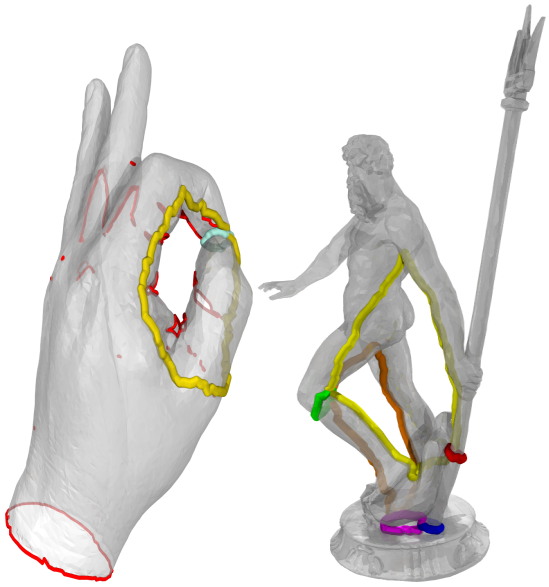

Figure 10. Left: Fundamental cycles on a manifold with boundaries (12,000 faces). Boundaries are shown in red. No special precautions are necessary to handle them. **Right:** Fundamental cycles on genus 3 Neptune model (34,000 faces).

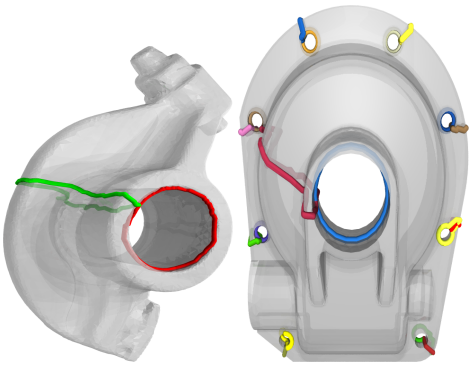

Figure 11. Fundamental cycles produced by our algorithm using the weights favoring minimum curvature direction, followed by simple cycle tightening. Both meshes have 21,000 faces.

Acknowledgements

This work has been partially supported by the NSF under grants CCF-0811809 and 0830403, and by the Office of Naval Research under grant N00014-08-1-1015. We would like to thank Uddipan Mukherjee and Ankit Gupta for their help preparing many figures in this paper. We also thank Intel Corporation for its disinterested support in the final stages of this work.

References

[BFC00] BENDER M. A., FARACH-COLTON M.: The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings* (2000), Gonnet G. H., Panario D., Viola A., (Eds.), pp. 88–94.

[CC07] CABELLO S., CHAMBERS E. W.: Multiple source shortest paths in a genus g graph. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2007), pp. 89–97.

[CCdVE*08] CHAMBERS E. W., COLIN DE VERDIÈRE É., ERICKSON J., LAZARUS F., WHITTLESEY K.: Splitting (complicated) surfaces is hard. *Comput. Geom. Theory Appl. 41*, 1-2 (2008), 94–110.

[CdVL07] COLIN DE VERDIÈRE É., LAZARUS F.: Optimal pants decompositions and shortest homotopic cycles on an orientable surface. *J. ACM 54*, 4 (2007), 18.

[CM07] CABELLO S., MOHAR B.: Finding Shortest Non-Separating and Non-Contractible Cycles for Topologically Embedded Graphs. *Discrete Comput. Geom. 37*, 2 (2007), 213–235.

[CMEH*03] COLE-MCLAUGHLIN K., EDELSBRUNNER H., HARER J., NATARAJAN V., PASCUCCI V.: Loops in reeb graphs of 2-manifolds. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry* (New York, NY, USA, 2003), ACM, pp. 344–350.

[DC91] DONALD B. R., CHANG D. R.: On the complexity of computing the homology type of a triangulation. In *FOCS* (1991), pp. 650–661.

[DE93] DELFINADO C. J. A., EDELSBRUNNER H.: An incremental algorithm for betti numbers of simplicial complexes. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry* (New York, NY, USA, 1993), ACM, pp. 232–239.

[DEG99] DEY T. K., EDELSBRUNNER H., GUHA S.: Computational topology. In *Advances in Discrete and Computational Geometry* (1999), American Mathematical Society, pp. 109–143.

[DLS07] DEY T. K., LI K., SUN J.: On Computing Handle and Tunnel Loops. In *CW '07: Proceedings of the 2007 International Conference on Cyberworlds* (Los Alamitos, CA, USA, 2007), IEEE Computer Society Press, pp. 357–366.

[DLSCS08] DEY T. K., LI K., SUN J., COHEN-STEINER D.: Computing geometry-aware handle and tunnel loops in 3D models. *ACM Transactions on Graphics 27*, 3 (2008), 1–9.

[Edm65] EDMONDS J.: Paths, trees, and flowers. *Canad. J. Math. 17* (1965), 449–467.

[EHP02] ERICKSON J., HAR-PELED S.: Optimally cutting a surface into a disk. In *Sym. Comp. Geom.* (New York, NY, USA, 2002), ACM Press, pp. 244–253.

[EIT*90] EPPSTEIN D., ITALIANO G. F., TAMASSIA R., TARJAN R. E., WESTBROOK J., YUNG M.: Maintenance of a minimum spanning forest in a dynamic planar graph. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1990), pp. 1–11.

[Epp03] EPPSTEIN D.: Dynamic generators of topologically embedded graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2003), pp. 599–608.

[EW05] ERICKSON J., WHITTLESEY K.: Greedy optimal homotopy and homology generators. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2005), pp. 1038–1046.

[Gal86] GALIL Z.: Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys 18*, 1 (1986), 23–38.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 355–361.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings ACM SIGGRAPH* (1997), ACM SIGGRAPH, pp. 209–216.

[GR05] GERSTEN S. M., RILEY T. R.: Some duality conjectures for finite graphs and their group theoretic consequences. *Proc. Edin. Math. Soc. 48*, 2 (2005), 389–421.

[GT83] GABOW H. N., TARJAN R. E.: A linear-time algorithm for a special case of disjoint set union. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing* (New York, NY, USA, 1983), ACM, pp. 246–251.

[GT91] GOODRICH M. T., TAMASSIA R.: Dynamic trees and dynamic point location. In *Proc. 23rd Symp. Theory of Computing* (New York, NY, USA, May 1991), ACM, pp. 523–533.

[GY03] GU X., YAU S.-T.: Global conformal surface parameterization. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 127–137.

[HT84] HAREL D., TARJAN R. E.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing 13*, 2 (1984), 338–355.

[KKT95] KARGER D. R., KLEIN P. N., TARJAN R. E.: A randomized linear-time algorithm to find minimum spanning trees. *J. ACM 42*, 2 (1995), 321–328.

[KR87] KIDWELL M. E., RICHTER R. B.: Trees and Euler Tours in a Planar Graph and its Relatives. *The American Mathematical Monthly 94*, 7 (1987), 618–630.

[LEF*98] LIANG J., EDELSBRUNNER H., FU P., SUDHAKAR P. V., SUBRAMANIAM S.: Analytical shape computation of macromolecules: I. Molecular area and volume through alpha shape. *Proteins 33*, 1 (October 1998), 1–17.

[LPVV01] LAZARUS F., POCCHIOLA M., VEGTER G., VERROUST A.: Computing a canonical polygonal schema of an orientable triangulated surface. In *Proc. 17th Annual Symposium on Computational Geometry* (New York, NY, USA, 2001), ACM, pp. 80–89.

[LYC*06] LEE T.-Y., YAO C.-Y., CHU H.-K., TAI M.-J., CHEN C.-C.: Generating genus-n-to-m mesh morphing using spherical parameterization. *Comput. Animat. Virtual Worlds 17*, 3-4 (2006), 433–443.

[MV80] MICALI S., VAZIRANI V. V.: An $O(|V|^{1/2}|E|)$ algorithm for finding maximum matching in general graphs. In *FOCS '80: Proceedings of the 21st Symposium on Foundations of Computer Science* (1980), pp. 17–27.

[Nat92] NATARAJAN B. K.: *On Generating Topologically Correct Isosurfaces from Uniform Samples*. Tech. Rep. HPL-91-76, Hewlett Packard Laboratories, May 5 1992.

[NGH04] NI X., GARLAND M., HART J. C.: Fair morse functions for extracting the topological structure of a surface mesh. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 613–622.

[PSBM07] PASCUCCI V., SCORZELLI G., BREMER P.-T., MASCARENHAS A.: Robust on-line computation of Reeb graphs: simplicity and speed. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 58.

[PSF08] PATANE G., SPAGNUOLO M., FALCIDIENO B.: Reeb graph computation based on a minimal contouring. *Shape Modeling and Applications: SMI* (June 2008), 73–82.

[Ree46] REEB G.: Sur les Points Singuliers d'une Forme de Pfaff Complétement Intégrable ou d'une Fonction Numérique. *Comptes Rendus de L'Académie des Sciences* (1946), 847–849.

[RT06] RILEY T. R., THURSTON W. P.: The absence of efficient dual pairs of spanning trees in planar graphs. *Electr. J. Combinatorics 13*, 1 (2006), N13.

[SKK91] SHINAGAWA Y., KUNII T. L., KERGOSIEN Y. L.: Surface Coding Based on Morse Theory. *IEEE Comput. Graph. Appl. 11*, 5 (1991), 66–78.

[Som58] SOMMERVILLE D. M. Y.: *An Introduction to the Geometry of N Dimensions*. Dover, 1958.

[SV88] SCHIEBER B., VISHKIN U.: On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing 17*, 6 (1988), 1253–1262.

[SV03] SZYMCZAK A., VANDERHYDE J.: Extraction of topologically simple isosurfaces from volume datasets. In *Proceedings of IEEE Visualization 2003* (October 2003), Turk G., van Wijk J., Moorhead R. J., (Eds.), pp. 67–74.

[Tar72] TARJAN R. E.: Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing 1*, 2 (1972), 146–160.

[WHDS04] WOOD Z. J., HOPPE H., DESBRUN M., SCHRÖDER P.: Removing excess topology from isosurfaces. *ACM Transactions on Graphics 23*, 2 (April 2004), 190–208.

[YJG07] YIN X., JIN M., GU X.: Computing shortest cycles using universal covering space. *Vis. Comput. 23*, 12 (2007), 999–1004.

[ZJLG09] ZENG W., JIN M., LUO F., GU X.: Canonical homotopy class representative using hyperbolic structure. In *IEEE International Conference on Shape Modeling and Applicat* (2009).