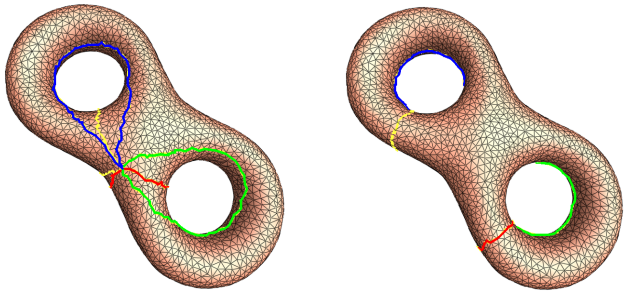
Computing Shortest Cycles Using Universal Covering Space

Xiaotian Yin, Miao Jin, Xianfeng Gu
Computer Science Department
Stony Brook University
Stony Brook, NY 11794, USA
{xyin, mjin, gu}@cs.sunysb.edu

Abstract

In this paper we generalize the shortest path algorithm to the shortest cycles in each homotopy class on a surface with arbitrary topology, by utilizing the universal covering space (UCS) in algebraic topology. In order to store and handle the UCS, we propose a two-level data structure which is efficient for storage and easy to process. For the shortest cycle algorithm and the UCS data structure, we showed some practical applications, such as topological denoise in geometric modeling, polygonal schema construction in computational topology and etc.



(a) (b)

Figure 1. Shortest cycles in different homotopy classes. (a) shows the shortest cycles passing through a common vertex. (b) shows the shortest cycles without the common vertex restriction.

1. Introduction

1.1. Problem Statement

The problem of finding shortest paths in graphs is a fundamental optimization problem with many applications, such as routing in networks, image segmentation in vision, surface segmentation in graphics, robot motion and navigation, speech recognition and VLSI design, etc. Algorithms with near-optimal efficiency, either in theory or in practice, are known for some problem variants, such as the single-source problem (see [3, 5, 21]) and the all-pairs shortest-path problem (see [4, 12, 20]).

On the other hand, computing the shortest cycles on a general surface is a more complicated problem. This is due to the fact that cycles on general surfaces belong to different homotopy classes, each class has its own shortest cycle and these shortest cycles cannot be smoothly deformed to one another without leaving the surface. For example, in figure 1.3 cycle c_1 and c_3 are in the same homotopy class, while c_0, c_1 and c_2 are in different homotopy classes. In particular, only c_0 can shrink to a point. In many practical problems, it is more useful to find the shortest cycles within certain homotopy classes instead of over all cycles on the surface.

In this paper, we studied the following fundamental problem:

General Shortest Cycle Problem: *Given a surface mesh M with arbitrary topology, find the shortest cycle in each homotopy class of M (Figure 1.1 (b)).*

In particular, we studied a restricted version of the above problem:

Restricted Shortest Cycle Problem: *Given a surface mesh M with arbitrary topology, find in each homotopy class the shortest cycle that passes through a given point p on M (Figure 1.1 (a)).*

This restricted problem can be solved more efficiently and gives direct intuition towards a solution to the general problem. Our algorithm is based on the universal covering space (see section 2.2); therefore we also investigated efficient data structures to handle the universal covering space.

1.2 Motivation

The solutions to the shortest cycle problem can benefit many important applications in graphics. By finding these shortest cycles, it is easy to compute the homotopy group of the surface, to which different topological operations can then be applied. In [6], a special set of homology basis curves is selected and the surface is sliced open along them to covert it to a geometry image. In the topological denoising [9] and topological simplification [23] work of Wood et al., the small handles are located and removed by slicing along the shortest cycles. In the works of surface parameterization [8], [7], the homotopy group bases are explicitly constructed. The results can be applied to mesh decomposition directly, to find cut curves on the surface with topological and geometric properties for metamorphosis [15], compression [13], shape revtrieval [24], texture mapping [18], etc.

1.3. Related Work

There are several possible approaches to solve the problem. We call the first approach *curvature flow method*. The active contour method [14] is widely used in computer vision. A planar curve on an image can be shrunk by moving each point towards its curvature center. The deformation process does not change the homotopy type of the curve. Later, in [17], geometric snakes, which are cycles on a surface, are computed from active contours on the corresponding parameter chart. By distorting the curve based on geodesic curvature, the curves will be deformed to geodesics with the same homotopy type. This approach has several disadvantages. First, the geodesics are at a local minimum; our goal is to find the global minimum. In addition, the iterative method is inefficient, and the computation of the geodesic curvature is unstable.

Another approach uses a computational topology method to construct the *universal covering space* (see Section 2) and lift the closed curves on the surface to curve segments in this space. We call this method the *universal covering space* method. Hershberger and Soneyink did some pioneering work on this problem [10] and [11] using this approach. However, their results only apply to *boundary-triangulated-2-manifolds* and cannot be used in our case because it assumes all vertices are on a boundary.

The shortest cycle problem has significant connections with other important problems in computational topology, such as constructing polygonal schema (cut a closed genus g surface to a canonical polygon with $4g$ edges), contractibility test and transformability test. Vetger and Yap sketched algorithms to construct canonical polygonal schema in [22]. Lazarus et al. improved the algorithm in [16]. The methods require refining the mesh; if it has n triangles, the result mesh may have $O(gn)$ faces, where g is the genus of the mesh. Based on [22], Schipper give an $O(g^2k + gn)$ time and space algorithm to detect the contractibility of curves in [19], where k is the length of the cycle. Dey and Schipper improved the algorithm to $O(n+gk)$ time by deriving a reduced polygonal schema in [2]. The method will lose topological and geometric information by merging some boundary vertices and cannot applied to our purpose. Dey and Guha solve the transformability problem in [1], where they abandon universal covering spaces and use combinatorial group theory to improve the complexity to $O(n+k_1+k_2)$. The method only reflects the topological information of the curves without considering geometric information; therefore, it can not be applied to our case either.

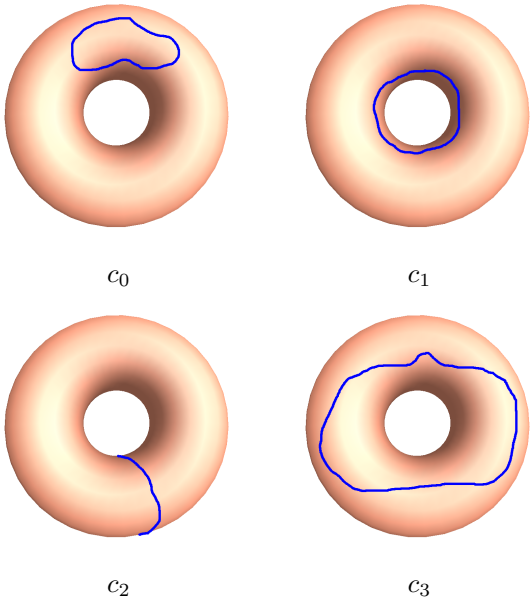


Figure 2. Cycles on a surface. c_0, c_1 and c_2 are in different homotopy classes; c_0 is contractible; c_2 and c_3 are transformable.

1.4. Contributions

To solve the shortest cycle problem, we proposed a general framework that utilizes the universal covering space (UCS) to transform the problem of finding shortest cycles into the problem of finding shortest paths. In order to avoid the exponential blowing up of the space required by naive construction of the UCS, we developed a space efficient data structure to handle UCS. We outline the contributions in the following.

1. We proposed a general algorithm to compute the shortest cycles (geometrically rather than combinatorially)

in each homotopy class. The algorithm can handle surface meshes with arbitrary topology, either with or without boundary.

2. We provided a storage efficient data structure to handle the universal covering space of a given mesh, which is scalable to meshes with complicated topology and/or of large size. It also provides a fundamental framework for other algorithms based on universal covering space.

3. We showed some potential applications of the shortest cycle algorithm and several extensions to our universal covering space data structure, which demonstrate the capability and flexibility of the framework.

This paper is organized as the following. In section 2 we introduce the concepts and theories involved in our algorithm. The details of the algorithm are presented in section 3, together with some experimental results. We outline applications and extensions in section 4 and conclude in section 5.

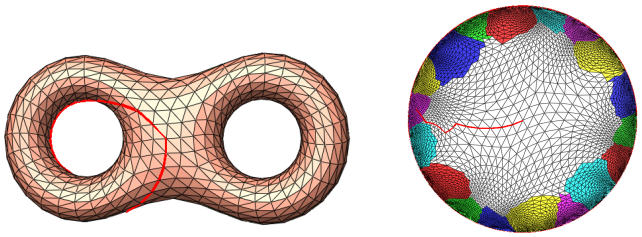
2. Theoretical Background

In this section we give an intuitive introduction to the concepts and theories involved in our algorithm. The concepts are explained both in the smooth setting and in the discrete setting.

2.1. Homotopy Classes

Intuitively, two closed curves are *homotopic* to each other on a given surface if and only if one can be smoothly deformed to the other without leaving the surface (such as c_1 and c_3 in Figure 1.3). homotopy is an equivalence relation, it classifies the set of closed curves on a given surface into a set of *homotopy classes*, where cycles in each class are transformable to one another while cycles in different classes are not. Under the operation of cycle product, all homotopic equivalence classes form the so called *fundamental group* of the given surface S, denoted as $\pi_1(S)$.

The same concepts can be defined on surface meshes similarly. Suppose γ is a cycle on mesh M, $\gamma = e_0, e_1, \cdots, e_n$, a face f is adjacent to γ, and the boundary of f is $\partial f = \bar{e}_1 + \bar{e}_2 + \bar{e}_3$, where $\bar{e}_1 = -e_i$. An *elementary transformation* of γ replaces e_i by $\bar{e}_2, \bar{e}_3$, and the result cycle is $\gamma = e_0, \cdots, e_{i-1}, \bar{e}_2, \bar{e}_3, e_{i+1}, \cdots, e_n$. Two cycles γ_1 and γ_2 are homotopic if there are finite elementary transformations to transform γ_1 to γ_2. Over this discrete definition of homotopy, the homotopy classes and fundamental group for meshes are defined in the same way as in the smooth setting.



(a) (b)

Figure 3. Cycle lifting. (a) is the original mesh, (b) is the flattend UCS. The red cycle in (a) is lifted to the red path in (b).

2.2. Universal Covering Space

Given a connected surface S, its *Universal Covering Space* (UCS) is defined as a pair $(\overline{S}, \pi)$, where $\overline{S}$ is a simply connected surface, π is a continuous transformation from $\overline{S}$ onto S such that for each point $q \in S$ there are multiple pre-images $p \in \overline{S}$, and each p has a neighborhood that via π is topologically equivalent to a neighborhood of q. The transformation π is called a *covering map*. Intuitively, $\overline{S}$ consists of multiple copies of S sewed together, covering S by multiple times by the covering map π.

For the interest of simplicity, we also use UCS to represent $\overline{S}$ (Figure 3 (c)). Each piece in the UCS is called a *fundamental domain* (Figure 3 (b)).

On the UCS $\overline{S}$ we can define transformation $f : \overline{S} \to \overline{S}$ among sheets. If such a transformation is a homeomorphism such that $\pi \circ f = \pi$, it is called a *deck transformation*. Every deck transformation corresponds to a homotopy class.

By the definition of covering map π, every cycle c passing through a base point p in M *lifts* to a path $\bar{c}$ (either closed or not), whose end points $\bar{p_0}$ and $\bar{p_1}$ are both pre-images of p. See Figure 2.1 for an example of cycle lifting. As we can see, $\bar{c}$ depends on the choice of the starting point $\bar{p_0}$ and the homotopy class of c. Intuitively, if a cycle c is homotopic to a point, then its lifting $\bar{c}$ is still a cycle, whose starting and ending points coincides at one pre-image of p; otherwise, $\bar{c}$ is a curve connecting two different pre-images of p.

In the discrete setting, we use mesh M and $\overline{M}$ to represent surface S and $\overline{S}$. The covering map is a surjective linear simplicial map between the vertex sets of $\overline{M}$ and M. The pair $(\overline{M}, \pi)$ forms the UCS of M. The lifting of curves can be defined accordingly in the discrete setting, which allows us to transform the problem of computing shortest cycles on M to the problem of computing shortest paths on $\overline{M}$.

3. Algorithms

The basic idea of our algorithm is to transform the problem of computing shortest cycles on M to the problem of computing shortest paths on $\overline{M}$ (a finite portion of the universal covering space of M).

In order to compute $\overline{M}$, we need compute a fundamental domain $\widetilde{M}_0$ by cutting the mesh M along a set of curves. Then we take $\widetilde{M}_0$ as the center copy and glue more copies $\{\widetilde{M}_i\}$ ($i \in [1..m]$) along the cutting segments to form $\overline{M}$. Suppose v is a vertex on M, and v_i is the pre-image of v in $\widetilde{M}_i$. Since each pair $(\widetilde{M}_0, \widetilde{M}_i)$ determines a homotopy class of M; the shortest path from v_0 to v_i for certain i corresponds to the shortest cycle through vertex v in a certain homotopy class. Thus we can solve the restricted shortest cycle problem by utilizing $\overline{M}$. To remove the restriction of passing through a fixed vertex, we just need to loop v through all vertices of M and keep the minimum-length shortest path for each pair $(\widetilde{M}_0, \widetilde{M}_i)$, which solves the general shortest path problem.

In brief, our algorithm goes through the following steps:

1. Compute a fundamental domain $\widetilde{M}_0$ (*section 3.1*).
2. Compute $\overline{M}$ by gluing multiple copies of the fundamental domain $\{\widetilde{M}_i\}$ ($i \in [0..m]$) with $\widetilde{M}_0$ as the center copy (*section 3.2*).
3. For a vertex v on M, and one of its pre-images $\overline{v_0}$ in $\widetilde{M}_0$, find the shortest paths on $\overline{M}$ from $\overline{v_0}$ to $\overline{v_i}$ for $i \in [1..m]$ (*section 3.3*).
4. repeat step 3 for each vertex v on M and keep the minimum-length shortest path connecting $\widetilde{M}_0$ and $\widetilde{M}_i$ for each $i \in [1..m]$ (*section 3.3*).

Since we only compute a finite portion of the universal covering space, the algorithm only outputs the shortest cycles for a finite number of homotopy classes rather than for all of them. But in practice, usually only certain homotopy classes are of interests and our algorithm suffices. Even if one is interested in all the classes, our algorithm can compute the bases of the fundamental group, which is finite.

As a note, the above algorithm pipeline can handle surfaces with arbitrary topology, either with boundary or without boundary.

In the following we will discuss the details of each step in the pipeline respectively. In particular, we will discuss the storage-efficient data structure for handling UCS in section 3.2.

3.1. Compute Fundamental Domain

A fundamental domain $\widetilde{M}$ is a topological disk that covers mesh M once. Figure 3 (b) shows a flattened fundamental domain for the kitten model. Intuitively, it can be obtained by cutting the mesh open along a certain set of curves, called *cut graph*, on the mesh. In our algorithm, we use a set of *homology bases* computed in [8] as the cut graph. by cutting, the cut graph in the original mesh will become the *cutting boundary* in the fundamental domain.

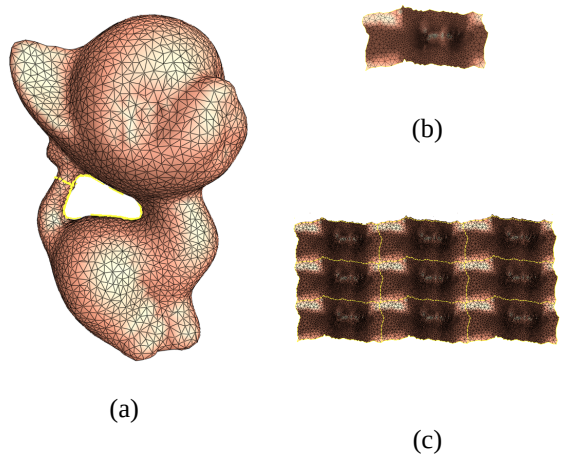


Figure 4. Fundamental domain and UCS. (a) is the original mesh with cut graph. (b) is the flattened fundamental domain, with four cutting segments $\{\widetilde{s_1}, \widetilde{s_2}, \widetilde{s_1}^{-1}, \widetilde{s_2}^{-1}\}$. (c) is a finite portion of the universal covering space (flattened onto the plane).

To facilitate later construction of the UCS, we partition the cutting boundary of the fundamental domain into *cutting segments*. For such segmentation, we need find the *junction vertices* in the cut graph whose degree is not two. These vertices will partition the cut graph into k simple curve segments s_i on M, each s_i will result in two cutting segments, $\widetilde{s_i}$ and $\widetilde{s_i}^{-1}$ on the cutting boundary of $\widetilde{M}$, where $\widetilde{s_i}$ and $\widetilde{s_i}^{-1}$ are called *dual segments*.

Here is the algorithm outline to compute a fundamental domain $\widetilde{M}$ and a set of cutting segments for a given mesh M.

1. Compute a set of homology bases (by [8]) as a cut graph Γ.
2. Cut M open along curves in Γ and get an open mesh $\widetilde{M}$.
3. Partition the cutting boundary of $\widetilde{M}$ by junction points in Γ to get the set of cutting segments $\{\widetilde{s_1}, \widetilde{s_1}^{-1}, \widetilde{s_2}, \widetilde{s_2}^{-1}, \ldots, \widetilde{s_k}, \widetilde{s_k}^{-1}\}$.

3.2. Compute Universal Covering Space with Storage-Efficient Data Structure

Constructing a finite portion of the universal covering space $\overline{M}$ is the key of the whole pipeline. The major difficulty here is to reduce the memory space taken by $\overline{M}$. In a naive manner, we can construct a mesh to represent $\overline{M}$ by gluing a set of fundamental domain copies. However, such a method will eat up the memory space quickly with increasing number of fundamental domain copies for a high genus surface.

In fact the storage requirement can be reduce dramatically. By further inspections we can see that each $\widetilde{M}_i$ has the same structure. It is not necessary to store such structure more than once in the universal covering space $\overline{M}$. Based on this fact, we proposed a two-level data structure to store $\overline{M}$ compactly.

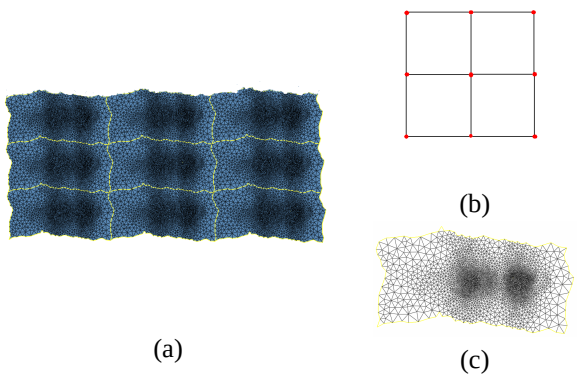


Figure 5. The two-level data structure for UCS. (a) is the flattened UCS. (b) is the high level graph $\mathcal{G}$, capturing the connectivity among pieces in the UCS. (c) is the low level graph, capturing the internal structure of each piece in the UCS.

The Data Structure At the low level of the UCS data structure, we reuse the mesh of one fundamental copy $\widetilde{M}_0$ to keep the local structure (Figure 3.2 (c)). At the high level, we need to construct an undirected graph $\mathcal{G}$ to capture the connectivity among the copies of fundamental domains (Figure 3.2 (b)). Each vertex v_i in $\mathcal{G}$ represent one fundamental domain copy $\widetilde{M}_i$, and is assigned with a unique id c_i to identify the fundamental domain copy this high level vertex represents. Further, there is an edge (v_i, v_j) if and only if $\widetilde{M}_i$ and $\widetilde{M}_j$ are glued together directly along some cutting segments.

Under such a data structure, each vertex of the universal covering space $\overline{M}$ can be identified by a global id (c_i, v_j), where c_i is the copy id of the resident fundamental domain $\widetilde{M}_i$ (i.e. the vertex id of the corresponding vertex in the high level graph $\mathcal{G}$), v_j is the vertex id within the fundamental domain. Each vertex not on any cutting segment owns a unique global id, while that on cutting segments will have multiple alias, one for each incident copy of fundamental domain according to $\mathcal{G}$. The correspondence among alias of the same vertex actually reflect the splitting of a given vertex in the cut graph, and can be obtained easily while we build the cutting segments in the previous step.

Constructing UCS Constructing a UCS using the two-level data structure is straightforward. Since we already have a fundamental domain from previous step, here we focus on constructing the high level graph $\mathcal{G}$. The following is the outline for this task.

1. Initialize $\overline{M}$ with the center copy of fundamental domain $\widetilde{M}_0$, initialize $\mathcal{G}$ with a single vertex c_0 that represents $\widetilde{M}_0$.
2. For $i \in [1..m]$ add a new copy $\widetilde{M}_i$ into $\overline{M}$ iteratively:
   (a) add a new vertex c_i into $\mathcal{G}$.
   (b) Glue $\widetilde{M}_i$ to the cutting boundary of current $\overline{M}$ along a maximum continuous set of cutting segments.
   (c) Whenever $\widetilde{M}_i$ is glued to another copy $\widetilde{M}_j$ in the previous step, add a new edge (c_i, c_j) in $\mathcal{G}$.
   (d) Update $\overline{M}$ and the cutting boundary of $\overline{M}$.
3. Output the high level graph $\mathcal{G}$.

Figure 3 (c) and Figure 2.1 (b) give examples of the flattened UCS we computed for a genus one model and a genus two model respectively.

Traversing UCS This two-level data structure not only saves memory space to store $\overline{M}$, but also allows fast traversal on $\overline{M}$. In brief words, traversing within a fundamental domain only needs the low level graph, while traversing across different domains involves the high level graph.

Now let's get into some details. Suppose we are travelling from vertex (c_i, v_j) to one of its neighbor vertices whose local id is $v_j{}'$. If the source vertex is not on any cutting segment, then the move is in the same fundamental domain $\widetilde{M}_i$, leading to target vertex $(c_i, v_j{}')$ directly. Otherwise, we can identify the right target vertex $(c_i{}', v_j{}')$ utilizing the alias correspondence of the source vertex, and make a move into a different fundament domain copy $c_i{}'$. As a note, each move in our UCS data structure is determined; there is no ambiguity.

3.3. Compute Shortest Cycles

As state at the beginning of the paper, the major problems we are studying are the restricted shortest cycle problem and general shortest cycle problem. Both of them can be solved using our universal coving space data structure.

As discussed in the previous section, given a mesh M we can build a finite portion of its universal covering space $\overline{M}$ by gluing a set of fundamental domain copies $\{\widetilde{M}_0, \widetilde{M}_1, \ldots, \widetilde{M}_m\}$. In order to compute the shortest paths in each homotopy class passing through a given vertex v on M, it is sufficient to find the shortest path between $\widetilde{v}_0$ and $\widetilde{v}_i$ for each $i \in [0..m]$, where $\widetilde{v}_i$ is the pre-image of v in fundamental domain $\widetilde{M}_i$.

Here we outline the subroutine for computing the shortest cycles passing through a given vertex v.

1. On the given $\overline{M}$, locate the set of pre-images of v, $\pi^{-1}(v) = \{\widetilde{v}_0, \widetilde{v}_1, \cdots, \widetilde{v}_m\}$, where $\widetilde{v}_i \in \widetilde{M}_i$.

2. For each $i \in [0..m]$, compute the shortest path $\widetilde{\gamma}_i$ connecting $\widetilde{v}_0$ and $\widetilde{v}_i$ on $\overline{M}$.

3. Project each $\widetilde{\gamma}_i$ back onto M, the projected image $\gamma_i = \pi(\widetilde{\gamma}_i)$ is the shortest cycle (passing through v) in one of the homotopy classes.

The above subroutine gives solutions to the restricted shortest cycle problem. In order to solve the general shortest cycle problem, we only need to loop v through every vertex of M, call the above subroutine repeatedly, keep track of the minimum-length $\widetilde{\gamma}_i$ for each i, and project back onto M to get the set of shortest cycles within different homotopy classes.

Figure 1.1 (a) shows a result of our restricted shortest cycle algorithm on a two hole torus model. Figure 1.1 (b) and Figure 4 show the results of the general shortest cycle algorithm on a two-hole torus model and a sculpture model respectively.

4. Extensions and Applications

Our algorithm of computing shortest cycles can be extended to solve related problems, such as building polygonal schema, topological denosing and etc. The storage-efficient data structure for universal covering space can also be utilized to handle other problems in computational topology, such as testing contractibility and transformability.

Polygonal Schema For a given surface mesh M, if we cut it open along a *canonical cut graph* where there is only one junction vertex in the cut graph, then we get a disk-like mesh, which is called a *polygonal schema*.

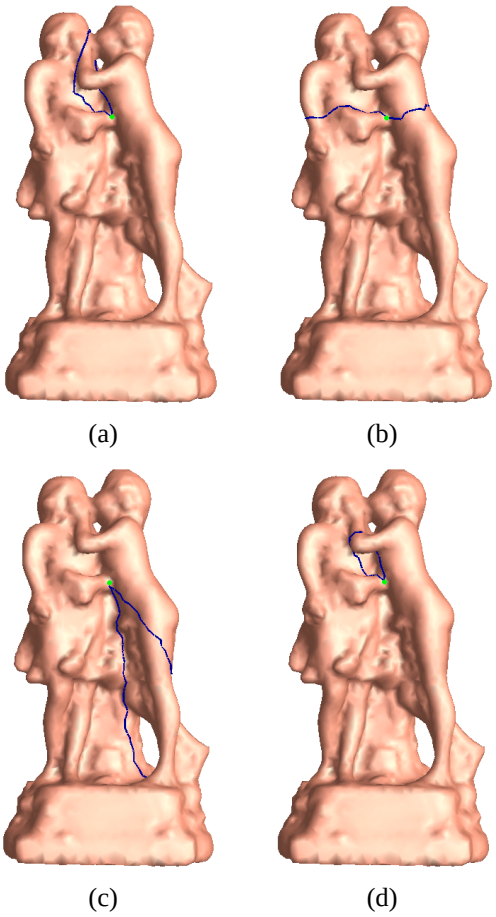


Figure 6. Shortest cycles through a common vertex. The marked shortest cycles in (a), (b), (c) and (d) are in different homotopy classes while passing through the same vertex (marked in green).

Our shortest cycle algorithm can help to compute a canonical cut graph. We start by an arbitrary cut graph, build the UCS and compute a set of shortest cycles passing through a given base point. We can choose a subset of these shortest cycles to form a homology bases, which is in turn a cut graph. Since all the cycles are the shortest ones, the cut graph is be very close to be canonical. If not, we can either perturb the cut graph to be canonical, or change the base point and retry. Upon the canonical cut graph is computed, we can construct the polygonal schema.

Topological Denoising With the advances of digital scanners and triangulation techniques, it is very easy to acquire surface meshes in nowadays. However, such meshes are usually very noisy, containing unexpected tiny handles. How to address and fix such topological noises is a chal-

lenging work.

Our shortest cycle algorithm provides an automatic method to find such tiny handles. We can compute the shortest cycles that winding each handle once, and order them by length. Those cycles at the lower end are good candidates for noise handles.

Contractibility and Transformability Given a curve on a surface, it is contractible if it can shrink to a point smoothly (for instance, cycle c_0 in Figure 1.3). Two curves are transformable to each other if one can be smoothly deformed to another without leaving the surface (for instance, cycle c_1 and c_3 in Figure 1.3). Our UCS data structure can be utilized to solve both of them.

For contractibility, we can build a finite portion of the universal covering space $\overline{M}$ for mesh M, lift the given cycle γ on M to a path $\overline{\gamma}$ on $\overline{M}$. If $\overline{\gamma}$ is also a cycle (i.e. its starting vertex coincides with the ending vertex), then γ is contractible; otherwise, γ is not contractible.

For contractibility test, we first bridge the two given cycles to form a bigger cycle, then check whether the bigger cycle is contractible or not. If and only if the answer is yes, the given cycles are transformable to each other.

5. Conclusion

In this paper the shortest cycle problems are studied in both the general case and a restricted case. We proposed an algorithm to compute the shortest cycles within each homotopy class on the general surface mesh with arbitrary topology, with or without boundary.

The algorithm utilizes the universal covering space of the given mesh, where a two-level data structure is proposed to store and handle the UCS. Such a data structure is not only compact in storage but also efficient for traversing, it can be used as a general framework for other algorithms based on UCS.

Finally We pointed some potential applications to the shortest cycle algorithm, as well as some extensions to the UCS data structure, which are all candidates for further exploration.

References

[1] T. K. Dey and S. Guha. Transforming curves on surfaces. *Journal of Computer and System Sciences*, 58:297–325, 1999.

[2] T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete and Computational Geometry*, 14:93–110, 1995.

[3] E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math*, 1:269–271, 1959.

[4] R. W. Floyd. Algorithm 97 (shortest path). *Communications of the ACM*, 5(6):345, 1962.

[5] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *JACM*, 34:596–615, 1987.

[6] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *SIGGRAPH*, 2002.

[7] X. Gu, Y. Wang, and S.-T. Yau. Multiresolution computation of conformal structures of surfaces. *Journal of Systemics, Cybernetics and Informatics*, 1(6), 2004.

[8] X. Gu and S.-T. Yau. Global conformal surface parameterization. In *ACM Symposium on Geometry Processing*, 2003.

[9] I. Guskov and Z. Wood. Topological noise removal. *Graphics Interface*, 2001.

[10] J. Hersberger and J. Snoeyink. Around and around: computing the shortest loop. In *Proceedings of the third Canadian Conference on Computational Geometry*, pages 157–161, 1991.

[11] J. Hersberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. In *Proceedings of the second Workshop on Algorithms and Data Structures*, 1991.

[12] D. B. Jason. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[13] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *Proceedings of ACM SIGGRAPH*, pages 279–286, 2000.

[14] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *International Journal of Computer Vision*, pages 321–331, 1988.

[15] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering. In *Proceedings of ACM SIGGRAPH*, pages 954–961, 2003.

[16] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Seventeenth Annual ACM Symposium on Computational Geometry*, 2001.

[17] Y. Lee and S. Lee. Geometric snakes for triangular meshes. *Computer Graphics Forum at Eurographics 2002*, 21(3):229–238, 2002.

[18] B. Levy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of ACM SIGGRAPH*, pages 362–371, 2002.

[19] H. Schipper. Determining contractiblity of curves. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 358–367, 1992.

[20] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 605–614, 1999.

[21] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *JACM*, 46:362–394, 1999.

[22] G. Vegter and C. Yap. Computational complexity of combinatorial surfaces. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 102–111, 1990.

[23] Z. Wood, H. Hoppe, M. Desbrun, and P. Schr

[24] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Comters and Graphics*, 2(5):733–743, 2002.