DOI: 10.1111/j.1467-8659.2011.02053.x
Pacific Graphics 2011
Bing-Yu Chen, Jan Kautz, Tong-Yee Lee, and Ming C. Lin
(Guest Editors)
Volume 30 (2011), Number 7

Optimized Topological Surgery for Unfolding 3D Meshes

Shigeo Takahashi[1], Hsiang-Yun Wu[1], Seow Hui Saw[1], Chun-Cheng Lin[2], and Hsu-Chun Yen[3]

[1]The University of Tokyo, Japan
[2]National Chiao Tung University, Taiwan
[3]National Taiwan University, Taiwan

Abstract

Constructing a 3D papercraft model from its unfolding has been fun for both children and adults since we can reproduce virtual 3D models in the real world. However, facilitating the papercraft construction process is still a challenging problem, especially when the shape of the input model is complex in the sense that it has large variation in its surface curvature. This paper presents a new heuristic approach to unfolding 3D triangular meshes without any shape distortions, so that we can construct the 3D papercraft models through simple atomic operations for gluing boundary edges around the 2D unfoldings. Our approach is inspired by the concept of topological surgery, where the appearance of boundary edges of the unfolded closed surface can be encoded using a symbolic representation. To fully simplify the papercraft construction process, we developed a genetic-based algorithm for unfolding the 3D mesh into a single connected patch in general, while optimizing the usage of the paper sheet and balance in the shape of that patch. Several examples together with user studies are included to demonstrate that the proposed approach works well for a broad range of 3D triangular meshes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems - mesh unfolding, topological surgery, genetic algorithms, papercraft models

1. Introduction

Unfolding 3D meshes into 2D papercraft models allows us to retrieve the corresponding 3D physical shapes from the 2D display to the real world. This technique would be very useful when we prepare miniatures of 3D scenes, such as architectural designs and urban plannings using the papercraft models. Furthermore, regardless of the recent development of rapid prototyping, constructing the 3D physical models from 2D papercraft patterns is by itself an interesting entertainment, especially for children and families to share pleasant experiences. A variety of methods have been developed recently for this purpose both in the fields of computational geometry and computer graphics.

Although existing techniques are effective, they usually decompose an input 3D model into a relatively large number of unfolded patches, including small pieces as shown in Figure 1(a). This is a serious problem in practice because we have to seek the correspondences between the boundary edges of different patches when merging them, or even worse, we have to fit a tiny piece, having a few faces only, to the remaining part of a 3D papercraft model tightly enough. Actually, facilitating simple search for the boundary edge matching is crucial for accelerating the construction of the corresponding papercraft model.

This paper presents a new heuristic approach for fully optimizing the 2D unfolding of an input 3D mesh. The key ideas of our approach are inspired by the concept of topological surgery, which allows us to encode the edge sequence on the boundary of the unfolding using a symbolic representation. Furthermore, in order to ease the papercraft construction process, we have developed a genetic-based algorithm for unfolding the 3D mesh into one single patch. This formulation allows us to construct the 3D papercraft model only through simple atomic operations for merging boundary edges of the 2D unfolding, where we can always find a pair of duplicated edges that are next to each other.

Our approach is distortion-free in the sense that we need not stretch and shrink unfolded patterns to construct the papercraft models, unlike most conventional approaches. Actually, unfolding a polyhedron into a single unfolded patch without any deformation is a well-known open problem [She75, DO05] and has intensively been studied so far

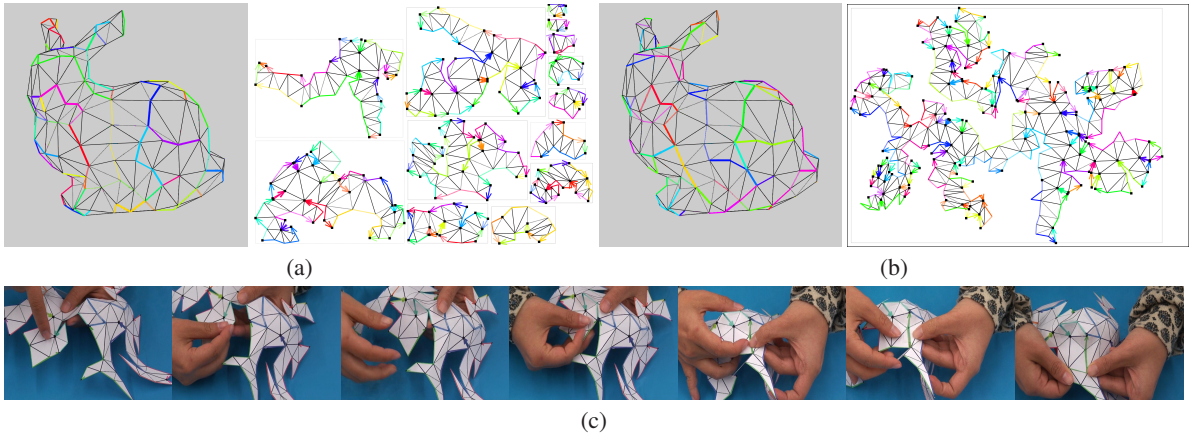


Figure 1: *Unfolding a bunny model (348 faces). (a) A previous approach [SP05] unfolds the mesh into many patches including small pieces. (b) Our approach converts the mesh into a single unfolded patch. (c) Steps for stitching boundary edges together one by one for papercraft construction. The arrows of the same color indicate the correspondences between duplicated edges together with their directions, while solid and broken lines on the patches represent mountain and valley folds, respectively.*

rather from a theoretical point of view. However, to our knowledge, the only available solution to this problem is to check all the possible combinations of edges to be cut over the input polyhedron until we find an acceptable solution, which may lead to combinatorial explosion. On the other hand, our genetic-based approach considerably accelerates the search for the overlap-free single connected patch by adaptively sampling the search space of mesh unfoldings. This is accomplished by classifying mesh vertices into hyperbolic and elliptic ones by referring to their surface curvatures, which lets us instantly reject local self-overlaps between a pair of neighboring faces without actually projecting them onto a 2D plane. As presented in Figure 1(b), our approach usually converts a 3D mesh into a single unfolded patch for 3D meshes with up to 500 faces, which can be considered as a size limit for hand-made papercrafts.

We assume that the input 3D mesh consists of triangular faces only, while this is not a hard constraint because we can easily partition a non-triangular face into a set of triangles. There are no restrictions on the topological types of the 3D meshes as long as they are orientable, which means that topological tori and their connected sums can be successfully unfolded without any extra cost. In our approach, we also assume that the unfolded patches will be printed on a single A3 sized sheet, to which the scale of the unfolded patches is automatically adjusted to be fit compactly.

Our approach begins by partitioning a given 3D mesh into a set of small unfolded patches. The unfolded patches are then stitched together along boundary edges one by one using a genetic algorithm. The genetic-based optimization ordinarily transforms the initial set of small patches to a single connected one, while we can optionally rearrange the configuration of the mesh unfolding to fully optimize the number of unfolded patches. The boundary edges of the unfolded patches are rendered using arrows having different colors, so that we can easily identify the correspondence between a pair of duplicated boundary edges together with their directions as shown in Figure 1(c). In addition, mountain and valley folds on the patches are explicitly rendered using solid and broken lines, respectively, in our implementation.

The remainder of this paper is organized as follows: Section 2 summarizes the previous approaches to 3D mesh unfolding. Section 3 describes the basic ideas of the proposed approach. The overall process is detailed from Section 4 to Section 6, where Section 4 describes how we decompose the input 3D mesh into a set of small unfolded patches, Section 5 introduces an algorithm for stitching together unfolded patches along boundary edges using a genetic algorithm, and Section 6 presents an optional stage for rearranging the mesh unfolding to fully minimize the number of unfolded patches. After demonstrating several examples of mesh unfoldings together with user studies in Section 7, Section 8 concludes this paper.

2. Related Work

Unfolding 3D polyhedra has been intensively studied in the field of computational geometry from a theoretical point of view [She75]. A typical frequently asked question is whether a polyhedron can be cut along its edges and unfolded onto a 2D plane without self-overlaps, while the only known solution to this question is to try all the possible combinations of edges to be cut. Readers can refer to recent surveys (e.g. [DO05]) for further details.

From a practical point of view, on the other hand, unfolding 3D models has been conducted after their shapes are fa-

vorably transformed. This type of approach has been well-studied especially in CAD applications since 1990s [PF95, Hos98], where the 3D shape is approximated as a set of developable surfaces that we can easily unroll onto a plane.

In computer graphics, Mitani et al. [MS04] presented a pioneering work where they successfully approximated an input 3D mesh with a set of triangular strips. This work was followed by several interesting extensions, which approximate the shape of a 3D mesh using a few types of developable surfaces [STL06, MGE07]. Mesh segmentation techniques [AKM*06] naturally let us unfold 3D meshes if we can allow shape distortion to some degree according to surface curvatures [JKS05, YGZS05]. Mesh parametrization techniques [FH05, SPR06] also helped us convert the 3D mesh onto a 2D plane because the techniques provide us with a one-to-one mapping between the 3D shape and 2D plane rather directly. In this framework, 3D meshes were flattened onto 2D with bounded distortion by introducing seams of minimal length [SCOGL02] and low visibility [SH02].

Distortion-free unfolding of 3D meshes is preferable in the sense that we can avoid crumpling sheets during the papercraft construction, while it still remains difficult since directly cutting out the given 3D mesh along its edges usually results in a large number of unfolded patches due to self-overlaps. There are several available mesh unfolding programs (e.g., http://www.javaview.de/services/unfold/) on the Internet, while they are limited to relatively simple 3D models with a small number of faces. Straub et al. [SP05] proposed an interesting approach in this category, where they unfolded an input 3D mesh by referring to a *minimum spanning tree* (MST) that covers the dual graph of the mesh, and then resolved the self-overlaps on the unfolded patches. However, the resulting mesh unfolding still retains a relatively large number of patches (cf. Figure 1(a)). Note that the input meshes are usually expected to be simplified to have up to 500 faces beforehand, so that we can construct the corresponding papercraft model within a certain period of time. We follow this strategy in our work by employing a mesh simplification technique [GH97].

3. Basic Ideas

This section describes the basic ideas of the proposed approach including the overview of our algorithm.

3.1. Topological surgery

Topological surgery [Mas80] is a mathematical formulation that allows us to classify closed surfaces according to their topological types. This can be accomplished by cutting out the closed surface along seams, and then encoding the appearance of its boundary edges, for example, in the counter-clockwise order. Note that the seams consist of *cut edges* and compose a spanning tree of such cut edges over the 3D mesh. Let us define a *boundary run* to be a sequence of cut edges

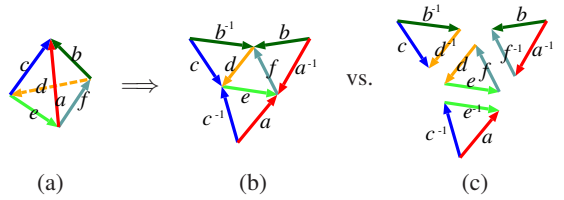


Figure 2: *Encoding the appearance of boundary runs. (a) A tetrahedron. (b) A single unfolded patch. (c) Multiple unfolded patches.*

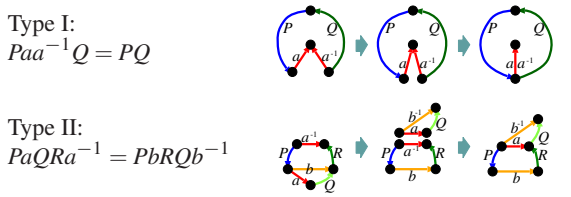


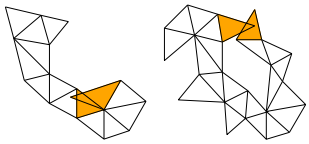
Figure 3: *Two fundamental transformations for orientable surfaces, where each symbol corresponds to a boundary run.*

bounded by the endpoints or branches on the spanning tree. Here, we denote the appearance of such a boundary run by a symbol, for example a, while we assign a^{-1} when we encounter the same run in the opposite direction. Figure 2(b) illustrates that the boundary of the unfolded tetrahedron is encoded by $aa^{-1}bb^{-1}cc^{-1}$ when we cut the tetrahedron along three edges a, b, and c as shown in Figure 2(a).

The formulation of topological surgery provides us with two fundamental transformations for the encoded boundary runs, as shown in Figure 3, if we limit ourselves to orientable surfaces. Type II transformation involves both cut and stitch operations for reordering boundary runs, and thus we cannot reduce the number of such runs. In contrast, Type I transformation monotonously reduces the number of boundary runs by stitching together a pair of duplicated runs that are next to each other while in opposite directions. Actually, this transformation is simple enough to acquire and thus has been employed in our approach as an atomic operation for composing the papercraft model. In practice, the classification theory guarantees that, through a sequence of the atomic operations, we can simplify the symbolic representation of boundary runs into aa^{-1} for a sphere, and $a_1b_1a_1^{-1}b_1^{-1}\cdots a_nb_na_n^{-1}b_n^{-1}$ for a surface with genus n. In this approach, we draw an arrow of different color on each boundary run so that we can easily find a pair of neighboring runs that can be stitched together (Figure 1(c)).

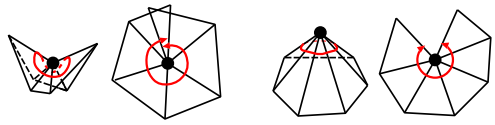
3.2. Unfolding a 3D mesh into a single unfolded patch

As described earlier, restricting our stitching operations to Type I of Figure 3 fully facilitates the construction of paper-



(a) (b)

Figure 4: *Self-overlaps on the mesh unfolding: (a) a local self-overlap and (b) a global self-overlap.*



(a) (b)

Figure 5: *(a) A hyperbolic vertex and (b) an elliptic vertex.*

craft models. For this purpose, we should unfold an input 3D mesh into a single connected patch, or a few patches at most. This is clear from Figure 2(c), where the tetrahedron is decomposed into multiple patches. In this case, we have to conduct different operations where we find the correspondence between a pair of boundary runs that are distributed to two distinct patches such as $\{d, d^{-1}\}$, $\{e, e^{-1}\}$, and $\{f, f^{-1}\}$. This usually increases the total time of papercraft construction considerably, especially for naïve users, as the number of unfolded patches becomes larger. However, if we can unfold the 3D mesh into a single connected component, we can easily construct the papercraft model through the sequence of atomic stitching operations of Type I only.

Indeed, Taubin et al. [TR98] studied how to unfold 3D meshes in the context of topological surgery. Nonetheless, their purpose is quite different from ours since they explored high compression ratio of the 3D mesh representation, by minimizing the number of bifurcations in the spanning tree of cut edges. Furthermore, they did not consider anything about the self-overlaps of the resulting 2D unfolding.

3.3. Classifying self-overlaps of mesh unfoldings

Our solution to this problem is to classify self-overlaps of mesh unfoldings into two types: a local self-overlap that causes an intersection between a pair of neighboring faces in the same connected patch (Figure 4(a)), and a global self-overlap where a pair of faces that are away from each other has a mutual overlap (Figure 4(b)). We hope to identify these self-overlaps before actually projecting the unfolded patches onto the 2D plane because we need geometric computation for that projection at the expense of relatively high computational cost. Actually, this is achievable especially for the local self-overlaps in our approach, by taking into account the surface curvature type at each vertex together with the number of cut edges incident to that vertex.

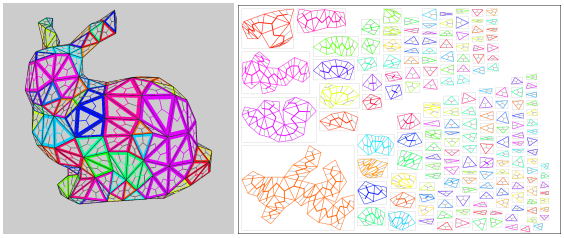
According to [BDE*03], we can classify the mesh vertices into *hyperbolic* vertices having negative Gaussian curvatures (Figure 5(a)) and *elliptic* vertices having positive Gaussian curvatures (Figure 5(b)). Here, the Gaussian curvature at each vertex is defined as the difference of 2π (i.e. 360 degrees) and the sum of the angles spanned by adjacent edges emanating from that vertex. It is obvious from Figure 5 that at each hyperbolic vertex we have to introduce at least two cut edges to avoid local self-overlaps because the sum of the corner angles around that vertex exceeds 2π, while we just require only one cut edge for an elliptic vertex.

The above observation implies that we can generally avoid local self-overlaps by updating the number of cut edges at each vertex whenever we split and merge the mesh surface. As for the global self-overlaps, there are no effective means of detecting them and thus we just transform the unfolded patches onto the 2D plane for detecting possible self-overlaps. However, as a preprocess, we roughly check the possible self-overlaps using the bounding boxes of the unfolded patches, which accelerates the necessary geometric computation significantly.

3.4. Processing pipeline

In fact, there are two extreme choices for seeking an overlap-free mesh unfolding: the first is to expand the spanning tree of cut edges adaptively over the 3D mesh, and the second is to compose the mesh unfolding by iteratively merging a set of single triangular faces. The first choice can never find such overlap-free unfoldings because we cannot evaluate how the current set of cut edges is close to the optimal solutions. The work by Straub et al. [SP05] is an example of this category. On the other hand, the second choice is unlikely to reach the optimal solutions due to excessive degrees of freedom in merging mesh faces. This observation suggests that we should explore the best tradeoff between the two extreme choices so that we can find an optimal unfolding within a certain period of time. This is why we decompose the input 3D mesh into a set of small unfolded patches first, and then stitch them together while avoiding undesirable self-overlaps. In practice, the success of this approach depends on how we control appropriate degrees of freedom in composing the final mesh unfolding, as will be discussed later.

Thus, the overall processing pipeline of our approach consists of three stages. The first stage of the pipeline is to seek a set of clustered faces by computing a set of spanning subtrees of dual edges. The clustered faces are then stitched together along boundary edges while avoiding self-overlaps in the second stage. As the final stage, we optionally rearrange the configuration of the unfolded patches so that we can avoid cases where the previous switching stage cannot fully optimize the number of connected patches. These three stages will be detailed in Sections 4-6.



(a) (b)

Figure 6: *Initial decomposition of a 3D mesh. (a) Spanning subtrees of dual edges together with face connectivity are drawn in different colors over the mesh. (b) The set of small unfolded patches, where 65% of the dual edges are disconnected. Gray line segments indicate stitchable edges.*

4. Mesh Decomposition

Our first task here is to decompose the input 3D mesh into a set of patches having a small number of faces.

4.1. Constructing spanning subtrees of dual edges

For obtaining unfolded patches having a small number of faces, we first extract the dual graph of the 3D mesh, and then assign an appropriate weight to each edge of the dual graph. Here, we borrow the *minimum perimeter* heuristic from [SP05], and define the weight value for the dual edge e as $w(e) = (l_{\max} - l)/(l_{\max} - l_{\min})$, by referring to the length of its corresponding primary edge l, where $l_{\min}$ and $l_{\max}$ denote the minimum and maximum lengths of all the primary edges, respectively. Finally, we compose spanning subtrees over the dual mesh by employing the dual edges whose weights are less than the predefined threshold. Note that the minimum perimeter weight assignment seeks the minimal total length of the seams (i.e., cut edges) over the mesh, and thus the seams are more likely to pass around its concave regions. This allows us to intuitively infer the shape semantics of the target mesh from the layout of the seams.

Selecting the appropriate threshold for the weights of dual edges is another important task because it controls appropriate degrees of freedom in composing the final mesh unfolding. Our experiments showed that we should try three different thresholds that disconnect 65%, 70%, and 75% of the dual edges in producing the initial configurations of small patches. As will be described in Section 7, we can usually obtain satisfactory results with one of these thresholds if the number of mesh faces is less than 500.

4.2. Resolving self-overlaps in an unfolded patch

We also would like to resolve possible global self-overlaps contained in each decomposed patch. This is accomplished by finding a pair of faces that intersect with each other on the 2D plane, then tracking the path between the faces on the unfolded patch, and finally cutting the patch along some edge on that path [SP05]. Note that this process also detects exceptional local self-overlaps that we cannot avoid just by counting the number of cut edges at mesh vertices. This exception occurs when the total sum of triangular corner angles exceeds 3π (i.e., 540 degrees), or the two cut edges span a very small angle around a hyperbolic vertex. In practice, the number of these exceptional cases is very small and we can usually reject the local self-overlaps by just managing the number of cut edges at each vertex. Figure 6(b) shows an initial set of small unfolded patches for the bunny model (Figure 6(a)), where 65% of dual edges are disconnected.

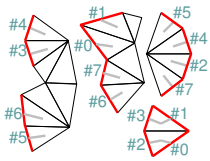


Figure 7: *Stitchable edges (in red), which are also indicated by gray line segments drawn from the center of the triangle.*

5. Stitching Unfolded Patches

Our next task is to stitch together a pair of unfolded patches one by one, so that we can minimize the number of unfolded patches while avoiding any self-overlaps.

5.1. Selecting stitchable boundary edges

First, we describe how to select boundary edges along which we try to stitch the corresponding pair of distinct patches. As described in Section 3.3, we can instantly identify boundary edges that are free of local self-overlaps by checking the number of cut edges at the corresponding end vertices together with their surface curvature types. We call such boundary edges *stitchable edges* in this paper. Figure 7 shows an example of unfolded patches and their stitchable edges where each edge is labeled by its corresponding ID. Of course, the stitchable edges still cause global self-overlaps (and a small number of exceptional local self-overlaps as described in Section 4.2), while we still significantly limit the number of edges for which we have to rigorously check the possible self-overlaps. Actually, this considerably reduces the search space for the optimized mesh unfolding, and thus can effectively accelerate the computation.

5.2. Encoding the order of stitchable edges

Now we focus on how to select a set of stitchable edges along which we stitch together the unfolded patches, in order to form an optimized layout of mesh unfolding. In our approach, we employ a *genetic algorithm* (GA) for seeking an optimal order of stitchable edges that successfully avoids

unwanted self-overlaps. Here, at first glance, one might expect the technique of *dynamic programming* to give a satisfactory solution, while a closer look reveals that various optimization issues in 3D mesh unfolding fail to comply with the so-called "optimal substructure" property in general, which is needed for dynamic programming to be applicable. On the other hand, the GA will provide better local search in the neighborhoods of the current solutions in the sense that the child solutions share good partial orders of stitchable edges with the parent solutions. In our setup, the order of stitchable edges is encoded as a chromosome with a sequence of edge IDs, as shown in Figure 8(a).

In the actual computation, we prepare an initial population of such chromosomes by generating randomly ordered edge IDs. However, we still need to rearrange the order of edge IDs, according to whether the corresponding stitchable edge can actually introduce a self-overlap-free combination of unfolded patches or not. Suppose that we have a chromosome as shown at the top of Figure 8(a). The edge ID in a white box corresponds to a stitchable edge along which we can successfully merge a pair of unfolded patches, while a gray box corresponds to a failure case due to self-overlaps. In our algorithm, we move the IDs of successful edges to the head of the chromosome while we push those of the failure cases to the back, as shown at the bottom of Figure 8(a). Rearranging the edge IDs is usually conducted when evaluating the fitness of the chromosome (Section 5.3), and helps us conduct further evolutional computation using crossover and mutation operations (Section 5.4). Note that a stitchable edge can become unstitchable due to local self-overlaps during the stitching stage, while this can be easily detected by faithfully updating the numbers of cut edges at the corresponding end vertices.

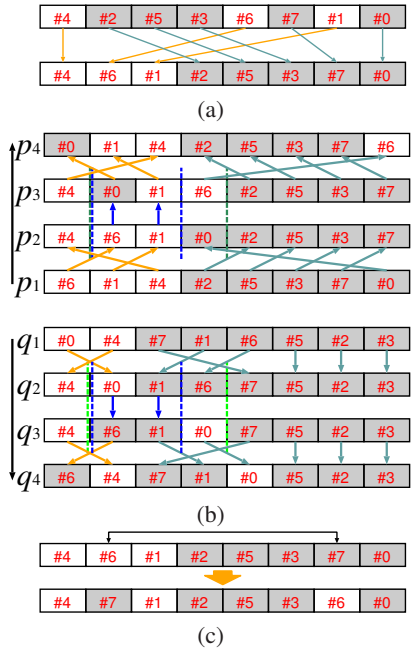


Figure 8: *Operations for chromosomes: (a) Reordering. (b) Crossover. (c) Mutation. Edge IDs in white and gray boxes correspond to successful and failure cases, respectively.*

5.3. Function for evaluating the fitness

Another important factor is to define an objective function that evaluates the fitness of each chromosome. In our approach, we formulate the objective function f to penalize several factors of mesh unfoldings and minimize it to find the best chromosome. In practice, f is defined as:

$$f = \lambda_p N_p + \lambda_l R_l + \lambda_m R_m + \lambda_b R_b. \quad (1)$$

Here, N_p is the number of unfolded patches, $R_l = F_l/F_t \in [0,1]$ where F_l represents the number of faces excluded from the largest patch and F_t represents the total number of faces, $R_m \in [0,1]$ is the relative ratio of the exterior margin around the mesh unfolding on the sheet, and $R_b = E_b/E_t \in [0,1]$ where E_b is the average number of edges between each pair of the duplicated boundary edges and E_t is the total number of boundary edges, respectively. λ_p, λ_l, λ_m, and λ_b are weight values for the four terms, respectively.

Since our primary objective here is to minimize the number of connected components, we first introduce the number as the first term of f. The second term is introduced due to the observation that the computation often converges to an equilibrium solution if multiple patches share almost the same number of faces. (See Section 7.3 also.) The third term is employed just for maximizing the relative area of the unfolded patches. Note that we use the approximate bin packing algorithm [IC01] in our approach, and freely change the scale of the unfolded patches so that we can minimize the area of the margin. The fourth term has been introduced to make each pair of duplicated boundary edges close to each other along the patch boundary. This certainly facilitates our papercraft construction since we are more likely to find pairs of corresponding boundary edges within a small neighborhood along the patch boundary. Note that we set the weight values as $\lambda_p = \lambda_l = 10\lambda_m = 100\lambda_b$ by default in our implementation, because minimizing the number of unfolded patches is our first priority and maximizing the coverage of the paper sheet is the next.

5.4. Crossover and mutation operations

Having prepared the initial population, we generally synthesize the next population by applying crossover and mutation operations to existing chromosomes. In our GA setup, these two operations are specifically designed for our problem so that we can produce better child chromosomes effectively.

Suppose that, before applying the crossover operation, we have two chromosomes p_1 and q_1, where the successful and

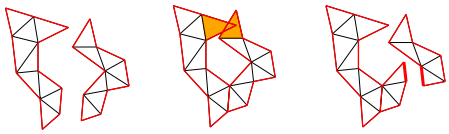


Figure 9: *Rearranging unfolded patches: (a) Two unfolded patches are initially separated. (b) The two patches are stitched regardless of self-overlaps. (c) Cut the stitched patch along a different edge to avoid self-overlaps.*

unsuccessful edge IDs have already been separated as shown in Figure 8(b). We then extract the intersections of the two sets of successful edge IDs, and rearrange p_1 and q_1 in a way that the edge IDs contained in the intersection (i.e. $\{\#4\}$) come first in the new chromosome p_2 and q_2 while the relative orders of edge IDs are preserved. The same can be applied to the two sets of unsuccessful edge IDs while common edge IDs (i.e. $\{\#2,\#3,\#5,\#7\}$) are pushed to the tail of each chromosome. Now we have remaining edge IDs that are contained neither in the successful nor unsuccessful intersections, in the centers of p_2 and q_2, which are bounded by green broken lines in the figure. Actually, we limit our crossover operation to those ranges, where the starting and ending positions for the actual crossover operation are randomly chosen in our genetic algorithm. This is more likely to produce child chromosomes that we have never encountered before, since we replace IDs of edges whose effects are not shared by the two parent chromosomes. In Figure 8(b), the crossover operation is performed to the sequences bounded by blue broken lines (from the second to third edge IDs), where #6 in p_2 is replaced with #0 in q_2 and #0 in p_2 is changed into #6 accordingly. The similar process is also applied to q_2 while guaranteeing that each edge ID appears only once in each chromosome. Here, p_3 and q_3 respectively represent the rearranged version of p_1 and q_1 to which we have applied our crossover operation. After this step, we apply inverse reordering operations to p_3 and q_3 so that we can retrieve the original order of edge IDs as in p_4 and q_4.

As for the mutation operation, we just randomly select one edge ID from each of the successful and unsuccessful edge ID sets, and then swap them as shown in Figure 8(c). This is because we may generate chromosomes we tested before just by swapping the edge IDs within the set of successful or unsuccessful edges. In our computation, the population will be updated in this way until the best score of the population will converge. Figure 1(b) exhibits the minimized layout of unfolded patches for the bunny model after this stage.

6. Rearranging Unfolded Patches

After a series of stitching operations based on the GA, we can usually obtain a single unfolded patch or otherwise a few unfolded patches at most. The reason why we may have more than one connected component is that the GA computation cannot fully optimize the configuration of mesh unfolding. However, the remaining unfolded patches can often be merged into a single patch by applying the local remerge and re-split operations. In practice, we have implemented Type II operation in Figure 3 for rearranging the partition of unfolded patches in our approach. Figure 9 illustrates how such a rearrangement operation will be carried out. First, we mark all the boundary edges as *visited*, which are colored in red as shown in Figure 9(a). We then find the boundary edge having the smallest weight value, and stitch together the corresponding pair of unfolded patches along that boundary edge, regardless of possible self-overlaps as shown in Figure 9(b). Finally, we compute the path between the faces having overlaps, cut the unvisited edge having the largest weight value, and mark the new cut edge as visited again, as shown in Figure 9(c). We continue this process until we can reduce the mesh unfolding into a single connected component.

7. Experimental Results

This section presents experimental results together with statistics and timing measurements for example papercraft models, followed by user studies and a discussion on our proposed approach. The source code of our approach together with a set of unfolded patterns is available at http://www.tak-lab.org/research/unfolding/.

7.1. Results

Our prototype system is implemented on a laptop PC with an Intel Core i7 CPU (2.67GHz, 4MB cache) and 8GB RAM, and the source code has been written in C++ using OpenGL and GLUT. In addition, we employed CGAL library for the mesh representation, Boost Graph Library for computing spanning subtrees, and GNU Scientific Library for laying out 2D unfolded patches on a paper sheet.

Figure 10 shows several examples of mesh unfoldings. In this computation, we prepared three initial sets of small unfolded patches for composing the final mesh unfoldings, where 65%, 70%, and 75% of dual edges were disconnected, respectively, as explained in Section 4.1. Table 1 shows the corresponding statistics and timings, where input meshes were transformed into a single unfolded patch in many cases just through the GA-based stitching stage, and in all the cases after the mesh unfoldings had been rearranged (cf. Section 6), as indicated by the arrows in Table 1. In our experiment, we could always find a single unfolded patch for any model having up to 500 faces if we tried the three initial mesh decompositions at least. For composing the final unfolded patch presented in Figure 10, we selectively employed a set of initial patches for each mesh as indicated in red in Table 1, so as to maximize the corresponding coverage

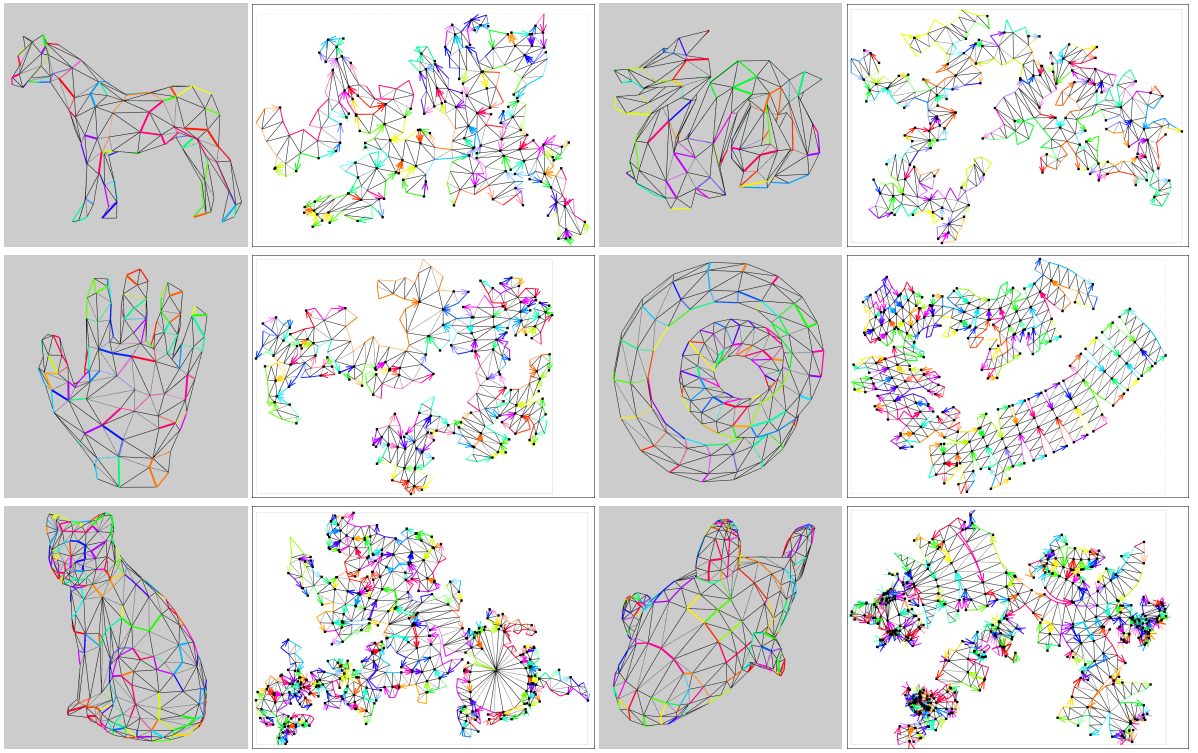


Figure 10: *Examples of mesh unfoldings: horse (312 faces), dragon (344 faces), hand (336 faces), knot (480 faces), cat (702 faces), and fish (950 faces) from top left to bottom right. In each case, cut paths over the 3D mesh are indicated on the left and its corresponding 2D unfolding is presented on the right. See Table 1 for the corresponding initial set of unfolded patches.*

Table 1: *Statistics and timing. f: Number of faces. n: Number of unfolded patches. t: Time for mesh unfolding (in seconds). r: coverage of the sheet (in percentage). The arrow means that we employed the optional rearrangement stage as a post-process.*

model	f	MST		Ours(65%)			Ours(70%)			Ours(75%)		
		n	r	n	t	r	n	t	r	n	t	r
bunny	348	13	36.6	1	31.8	30.9	1	19.0	20.5	1	40.8	27.6
horse	312	12	35.2	1	14.8	23.8	1	87.9	31.8	1	24.4	23.3
dragon	344	13	27.1	1	48.6	26.8	1	16.4	20.0	1	14.2	21.1
hand	336	18	32.5	1	54.3	30.9	2→1	18.3	20.4	1	24.0	16.6
knot	480	14	22.4	1	86.5	32.2	1	42.9	13.2	1	33.7	22.2
cat	702	28	32.2	1	332.2	32.0	4→1	202.3	14.1	2→1	244.5	20.1
fish	950	31	30.7	3→1	785.5	21.4	3→1	227.4	31.3	2→1	463.4	24.4

of the sheet. In practice, we can accomplish almost the same coverage as that of the conventional MST-based method on average. The computation time basically increases as the number of mesh faces becomes large, while the time also depends on the variation in the surface curvature of that mesh. For example, articulated models such as the horse, dragon, and hand meshes are harder to unfold because they have high negative curvatures around the joint parts. Note that in our GA computation, each population consists of 64 chromosomes and, at each generation update, half of them will be replaced with those obtained by crossover and mutation operations, where the probabilities of crossover and mutation are set to be 0.9 and 0.1, respectively.

Figure 11 exhibits some of these models assembled as papercrafts by hand together with the construction times. When compared with existing techniques, these construction times are significantly reduced to an acceptable period of time, with the help of a commercially available cutting plotter. We also equipped our system with an interface for specifying a series of edges as a seam in the final mesh unfolding by controlling the weights of the corresponding dual edges.

bunny	horse	dragon	hand	knot
34:01	41:57	55:46	33:55	46:13

Figure 11: *Hand-made papercraft models and construction times (min:sec).*

Table 2: *Timing of the first user study (min:sec). "F" and "M" represents female and male participants, respectively.*

	MST	Ours		MST	Ours		MST	Ours
A(M)	33:55	25:52	D(M)	19:19	19:40	G(F)	18:01	14:53
B(M)	13:52	10:31	E(M)	19:54	18:15	H(M)	18:15	14:10
C(F)	17:37	15:56	F(M)	17:58	16:41			

7.2. User studies

We conducted a user study to validate the effectiveness of our approach. We asked 8 participants (Participants A-H, aged 18-30) to construct a simplified bunny papercraft model (128 faces). We provided 6 distinct unfolded patches obtained by the conventional MST-based approach [SP05] (labeled "MST"), and a single unfolded patch generated using our approach (labeled "Ours"). Table 2 shows the results of this user study, where almost all the participants gave us positive responses to our ideas for providing a single unfolded patch in order to facilitate the papercraft construction, and visualizing correspondence between a pair of boundary edges using arrows of different colors. Note that Participants A-D tackled the multiple patches first while Participants E-H started with the single patch, in order to eliminate possible influence caused by learning effects. The construction time for the single unfolded patch was shorter than that for the multiple patches by 171.6 seconds on average.

We also conducted an additional user study by recruiting 6 additional participants (Participants I-N, aged 22-30) and asking them to construct the bunny model from a single unfolded patch (labeled "Ours") and those where hints on the order of stitching boundary edges were printed (labeled "Hint"). In this case, we ordered boundary runs so that the user can stitch the sharp edges at earlier stages and end by merging flat edges. This successfully reduced the total papercraft construction times by 167.7 seconds of the participants (I-N) on average as shown in Table 3, since the hints helped them start with the ears and face of the bunny model and then end with the back and bottom. Note that Participants I-K tackled the single patch without hints first while Participants L-N started by following the hints again.

Finally, we asked a 9-year-old boy to construct the same bunny model. He spent 23:53, 21:38, and 15:43 (min:sec) for the 3 different unfoldings ("MST", "Ours", and "Hint"), respectively. This means that our atomic operations for stitching boundary edges together with the hints fully facilitated the child to perform the papercraft construction.

Table 3: *Timing of the second user study (min:sec). "F" and "M" represents female and male participants, respectively.*

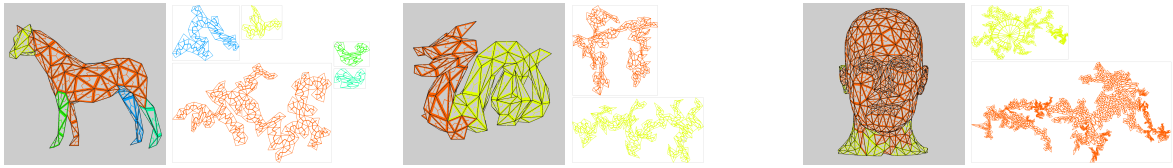
	Ours	Hint		Ours	Hint		Ours	Hint
I(M)	20:50	17:51	K(M)	22:15	17:00	M(F)	20:59	19:22
J(M)	16:45	15:20	L(M)	14:20	10:30	N(M)	23:40	22:00

7.3. Discussion

As described in Section 4.1, the quality of the papercraft patterns fairly depends on the initial set of small patches. For example, multiple patches obtained by the conventional MST-based approach [SP05] do not provide enough degrees of freedom in composing a single unfolded patch as shown in Figure 12(a). We also tested different strategies such as a weighted perfect matching [GE04], while we learned that our scheme based on spanning subtrees of dual edges with the minimum perimeter heuristic is the best. This is because the heuristic naturally inserts two or more cut edges at each hyperbolic vertex in concave regions, and thus avoids introducing unnecessary cut edges around elliptic vertices for better control of the degrees of freedom.

Another problem is to appropriately define the objective function that evaluates the fitness of each chromosome in the genetic-based composition of mesh unfolding. Actually, the second term in Eq. (1) plays an important role in our approach. Without this term, the genetic-based optimization is often trapped in an equilibrium state where multiple patches share almost the same number of faces (Figure 12(b)). Tweaking the probabilities of crossover and mutation operations is also important to stabilize the optimization. We tested different sets of probabilities for the two operations and learned that our choice of 0.9 and 0.1 for the probability of crossover and mutation operations, respectively, provides feasible evolutional computation in the sense that the choice explores child chromosomes effectively in the neighborhood of the current chromosome. In practice, changing these two probabilities fails to compose a single unfolded patch or degrades the coverage of the sheet at least.

Indeed, we often fail to find a single unfolded patch for a mesh having more than 1,000 faces (Figure 12(c)), because the degree of difficulty for unfolding 3D meshes mainly depends on the number of mesh faces. A large-sized population in GA is more likely to provide a better solution in these difficult cases while it increases the computation time on the other hand. Adjusting such tradeoff between the population size and computation efficiency adaptively according to the input 3D mesh is an important technical issue. However, as described earlier, we can basically limit the number of faces up to approximately 500 so that we can construct the corresponding papercraft model within a certain period of time.



(a) (b) (c)

Figure 12: *Failure cases where each unfolded patch is drawn in a different color. (a) We cannot compose a single patch from multiple ones obtained by the MST-based approach. (b) Our genetic-based optimization is trapped in a equilibrium state without the second term of the objective function. (c) We fail to unfold the mannequin model (1,376 faces) into a single connected patch.*

8. Conclusion

This paper has presented a new heuristic approach to transforming 3D meshes into an optimized layout of 2D unfoldings. The key ideas have been inspired by the concept of topological surgery, which allows us to construct papercraft models through atomic operations of stitching together boundary edges of the mesh unfolding. For this purpose, we have developed a genetic-based algorithm for optimizing the number of 2D unfolded patches together with the coverage of the paper sheet and the distance between each pair of boundary edges around the 2D unfolding. Several experimental results together with user studies and a discussion on our framework are included to justify the present approach.

Acknowledgements

We are grateful to all the participants for taking part in the user studies. This work has been partially supported by JSPS under Grants-in-Aid for challenging Exploratory Research No. 23650042, NSC 98-2218-E-009-026-MY3, Taiwan, and NSC 97-2221-E-002-094-MY3, Taiwan.

References

[AKM*06] ATTENE M., KATZ S., MORTARA M., PATANE G., SPAGNUOLO M., TAL A.: Mesh segmentation - a comparative study. In *Proc. Shape Modeling and Applications 2006* (2006), pp. 14–25.

[BDE*03] BERN M. W., DEMAINE E. D., EPPSTEIN D., KUO E. H.-S., MANTLER A., SNOEYINK J.: Ununfoldable polyhedra with convex faces. *Computational Geometry Theory and Applications 24*, 2 (2003), 51–62.

[DO05] DEMAINE E. D., O'ROURKE J.: A survey of folding and unfolding in computational geometry. In *Combinatorial and Computational Geometry*. Cambridge Univ. Press, 2005, pp. 167–211.

[FH05] FLOATER M. S., HORMANN K.: Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modeling*, Dodgson N. A., Floater M. S., Sabin M. A., (Eds.). Springer, 2005, pp. 157–186.

[GE04] GOPI M., EPPSTEIN D.: Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum 23*, 3 (2004), 371–379.

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

[Hos98] HOSCHEK J.: Approximation of surfaces of revolution by developable surfaces. *Computer-Aided Design 30*, 10 (1998), 757–763.

[IC01] IGARASHI T., COSGROVE D.: Adaptive unwrapping for interactive texture painting. In *Proc. Symp. Interactive 3D Graphics 2001* (2001), pp. 206–216.

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Computer Grahics Forum 24*, 3 (2005), 581–590.

[Mas80] MASSEY W. S.: *A Basic Course in Algebraic Topology*. Springer, 1980.

[MGE07] MASSARWI F., GOTSMAN C., ELBER G.: Papercraft models using generalized cylinders. In *Proc. Pacific Grahics 2007* (2007), pp. 148–157.

[MS04] MITANI J., SUZUKI H.: Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graphics 11*, 3 (2004), 259–263.

[PF95] POTTMAN H., FARIN G.: Developable rational Bézier and B-spline surfaces. *ACM Trans. Graphics 12*, 5 (1995), 513–531.

[SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *Proc. IEEE Visualization 2002* (2002), pp. 355–362.

[SH02] SHEFFER A., HART J. C.: Seamster: Inconspicuous low-distortion texture seam layout. In *Proc. IEEE Visualization 2002* (2002), pp. 291–298.

[She75] SHEPHARD G. C.: Convex polytopes with convex nets. In *Mathematical Proc. Cambridge Philosophical Society* (1975), vol. 78, pp. 389–403.

[SP05] STRAUB R., PRAUTZSCH H.: Creating optimized cut-out sheets for paper models from meshes. In *Proc. SIAM Conf. Geometric Design and Computing 2005* (2005).

[SPR06] SHEFFER A., PRAUN E., ROSE K.: Mesh parameterization method and their applications. *Foundations and Trends in Computer Graphics and Vision 2*, 2 (2006).

[STL06] SHATZ I., TAL A., LEIFMAN G.: Paper craft models from meshes. *The Visual Computer 22*, 9 (2006), 825–834.

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *ACM Trans. Graphics 17*, 2 (1998), 84–115.

[YGZS05] YAMAUCHI H., GUMHOLD S., ZAYER R., SEIDEL H.-P.: Mesh segmentation driven by gaussian curvature. *The Visual Computer 21*, 8–10 (2005), 659–668.