Edge-Compositions of 3-D Surfaces

Cynthia Sung*, **Erik D. Demaine, Martin L. Demaine, Daniela Rus**

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

Email: {crsung, edemaine, mdemaine, rus}@mit.edu

ABSTRACT

Origami-based design methods enable complex devices to be fabricated quickly in plane and then folded into their final 3-D shapes. So far, these folded structures have been designed manually. This paper presents a geometric approach to automatic composition of folded surfaces, which will allow existing designs to be combined and complex functionality to be produced with minimal human input. We show that given two surfaces in 3-D and their 2-D unfoldings, a surface consisting of the two originals joined along an arbitrary edge can always be achieved by connecting the two original unfoldings with some additional linking material, and we provide a polynomial-time algorithm to generate this composite unfolding. The algorithm is verified using various surfaces, as well as a walking and gripping robot design.[1]

1 Introduction

Today's engineering designs are limited by practical considerations of their fabrication. Although recent advancements in machining practices and 3-D printing technology have produced significant speedup in manufacturing of mechanical structures, these methods are still costly and time consuming when compared with planar fabrication alternatives [2]. Origami-based design methods aim to augment existing 2-D fabrication techniques with folding algorithms to allow rapid fabrication of 3-D structures. Fabricating structures in plane and then folding them into their final shape will allow complex devices to be created more quickly and efficiently, providing engineers with greater opportunities to prototype, test, and refine their designs.

Although folding in engineering design has existed in sheet metal products for decades [3], it is only recently being applied to electromechanical devices. Work in foldable circuits [4–6] has renewed interest in origami-inspired design methods.

*Corresponding author: Address: MIT CSAIL, 32 Vassar St, 32-376, Cambridge, MA 02139, USA

[1] Some material in this paper has been adapted from [1]

The ability to integrate sensors and circuitry directly into the physical body of a device has enabled rapid fabrication of fully functional robots [2, 7, 8]. In addition, folding as a vehicle for physical change has given rise to machines with customizable shape [6] and transformable structures that support rapid precise assembly of small devices [9].

Despite these successes, the design of folded structures has always been performed by humans and is often a long, iterative, and application-specific process. We present an algorithm that will introduce some automation into the design process. More specifically, we are interested in automatically composing multiple designs together so that the end product has the combined functionality of the originals. We take a primarily geometric approach and consider only the shape of the folded structure. Figure 1 illustrates the problem addressed. Given two folding patterns, in this case a walking robot and a gripper, our algorithm automatically generates a *composite folding pattern* for a walking robot with a gripper on one end.

figure 1

This paper addresses *edge-compositions*, compositions involving two surfaces connected at one edge via hinge joint. For ease of assembly of the final product, we require that the folding pattern be one piece. In addition, for structural reasons, it is desirable for the composite folding pattern to contain the original folding patterns, rather than for it to be a new unfolding. This is because in a folded state, cut edges, edges corresponding to edges on the boundary of an unfolding that have been glued together, are mechanically weaker than folds. Cutting along an edge that will be subjected to large stresses may drastically weaken the final product. We assume that the two inputted folding patterns satisfactorily perform their intended functions, and therefore require that the composite folding pattern contain the original unfoldings in their entirety as subsets.

Related Work

The problem of finding edge-compositions of unfoldings is related to that of edge-unfolding of polyhedra from the field of computational geometry (See [10] for a survey of results). Edge-unfolding involves flattening a polyhedron by cutting along some of its edges and unfolding the rest. The resulting planar figure should be a simple, non-overlapping polygon. The traditional problem statement for edge unfolding requires that every face be covered exactly once and that no extra material be added, leading to such results as [11] for which an unfolding does not exist. To ensure that an unfolding of an edge-composition can always be found, we allow the addition of extra material as long as it can be tucked away against an existing face.

In this sense, our problem is more similar to that studied by Tachi [12,13], where an arbitrary polyhedral surface is folded from a 2-D sheet of paper. Faces of the polyhedral surface are positioned on the 2-D plane, then excess material is tucked away to bring neighboring faces together. However, this process requires that neighboring faces on the polyhedral surface be placed adjacent in the plane. In contrast, we would like to keep our original fold patterns intact, and the faces touching the hinged edge may not always be able to be placed close to each other. Furthermore, Tachi's work forbids cuts, requiring that the unfolding be a convex polygon, and thus may be less efficient in terms of material usage. Finally, tucks protrude perpendicularly from the final resulting polyhedral surface, although theoretically they could be crimped to arbitrarily small length. If the surfaces produced are all static, this is not a major cause of concern. Since we would like to accommodate transformable folded surfaces, whose fold angles can change, we require tucks that fold flat.

The idea to generate an unfolding by combining simple surfaces into more complex ones is similar to Mitani's work [14, 15], in which rotational symmetry of the goal surface allows the surface to be decomposed into slices, each of which can be achieved by the same fold pattern. Similarly, Cheng and Cheong [16] decompose a surface into horizontal slices and construct its unfolding one level at a time. Both of these algorithms are limited in the types of surfaces that they can produce, and, like [13], the produced surfaces are static. Edge-compositions offer greater potential in the types and the degrees of freedom of resulting folded surfaces.

In sheet metal design, several efforts have also been made to automate the design process. For example, [17] proposes a multi-stage heuristic for generating a sheet metal part that covers all user-specified regions while avoiding user-specified obstacles. This is done by adding bridges between pairs of regions until every region is covered and the part is connected. In [18, 19], the authors solve a similar problem but enforce manufacturability of the part by generating only designs that can be produced using a set of allowed sheet metal operations. In both of these cases, the user inputs only constraints on what should or should not be covered, but the actual shape of the part is left to the algorithm, allowing greater flexibility to reduce cost and manufacturing time. Wang [20] considers the case where the desired shape of the sheet metal part is entirely known. When the shape is not manufacturable as a single piece, then it is decomposed into simpler manufacturable components that can be assembled via welding, riveting, etc. Wang does not attempt to combine the components into one sheet, again for ease of manufacturing.

With the difference in goals between the engineering and computational geometry communities comes a difference in modeling: where folding algorithms generally assume an ideal paper, a zero-thickness material that does not stretch, sheet metal designers specifically adjust for material thickness and deformations that occur when bending [21]. Since foldable electronics typically use a thin material, especially at joints, we use the ideal paper model.

Our Contributions

This paper considers edge-compositions of folded structures. Our main result is the following:

Any edge-composition of two folded surfaces has a one-piece non-self-intersecting unfolding consisting of 1) the unfoldings of the two original folded surfaces connected by 2) a bridge of linking material. (see Theorem 1)

We provide a polynomial-time algorithm for generating the composite unfolding and experimental evaluation across various input surfaces.

The remainder of this paper is structured as follows. Section 2 introduces necessary notation and states the problem being addressed. Section 3 gives the main insight behind generating composite fold patterns, and Section 4 provides the algorithm in greater detail. Section 5 contains the results of the algorithm for various edge-compositions. Sections 6 and 7 summarize the contributions of this work and provide directions for future study.

2 Definitions and Problem Statement

A *polygon* P is a planar figure topologically equivalent to a disc and bounded by a closed non-self-intersecting path composed of a finite number of line segments. This path is called the *boundary* ∂P of P, and the area enclosed is its interior $\mathring{P}$. The line segments on the boundary are *edges*, denoted $E(P) = \{e_1^P, e_2^P, \ldots\}$, and the points where two edges meet are *vertices*, denoted $V(P) = \{v_1^P, v_2^P, \ldots\}$.

A *polyhedral complex* Q is a union of a finite set of polygons such that the intersection of any two polygons, if nonempty, is an edge or a vertex of each. The polygons making up Q are called its *faces*. The vertices and edges of Q are the vertices and edges of its faces, and are denoted by $V(Q) = \bigcup_{P \in Q} V(P)$ and $E(Q) = \bigcup_{P \in Q} E(P)$ respectively.

A *folded state* of a polygon P is a mapping $\phi_F : P \to \mathbb{R}^3$ where the restriction map $\phi_F : \mathring{P} \to \mathbb{R}^3$ is isometric and noncrossing. We say that a polyhedral complex Q can be *unfolded* into P if Q is the image of a folded state $\phi_F(P)$ (see Fig. 2). The folded state $\phi_F(P)$ can also be represented as the union of polygonal faces. Every edge e^ϕ of the folded state that is not on the boundary corresponds to a *fold* $f = (e^P, \theta)$, where the *fold line* e^P is the line segment on P that corresponds to e^ϕ and the *fold angle* θ is equal to the dihedral angle between the faces of $\phi_F(P)$ sharing e^ϕ (Fig. 3). Positive fold angles correspond to what are commonly termed "valley folds," and negative fold angles are "mountain folds." The figures throughout this paper denote valley folds with dashed segments and mountain folds with dash-dotted segments. We denote the set of folds $F = \{(e_1^P, \theta_1), (e_2^P, \theta_2), \ldots\}$. The polygon P and these folds together comprise an *unfolding* (P, F) of Q, and they uniquely define the folded state $\phi_F(P)$. We call the exterior of P, $\mathbb{R}^2 \setminus P$, the *free space*.

figure 2

figure 3

We allow the following rigid transformations of an unfolding (P, F) in the plane:

- translation by a vector $\mathbf{v}$: P and all edges in F are translated by $\mathbf{v}$
- rotation by an angle α around a point $\mathbf{x}$: P and all edges in F are rotated by α about $\mathbf{x}$
- reflection: P and all edges in F are reflected over the x-axis. All fold angles in F are negated.

Then the problem we would like to solve is as follows.

Problem 1. *Given two polyhedral complexes Q_1 and Q_2 with unfoldings (P_1, F_1) and (P_2, F_2), and two edges $e^{Q_1} \in E(Q_1)$ and $e^{Q_2} \in E(Q_2)$, find an unfolding (P_3, F_3) such that*

1. *(P_3, F_3) is the unfolding of the union of Q_1 and Q_2 translated and rotated so that e^{Q_1} is coincident to e^{Q_2}, and*
2. *(P_3, F_3) contains translated, rotated, and/or reflected instances of (P_1, F_1) and (P_2, F_2) as subsets.*

The selected edges e^{Q_1} and e^{Q_2} act as a hinge in the combined surface. Informally, hinge joints are folds whose fold angles are not fixed but rather can take a range of values. In order for a solution to Problem 2 to make sense, the range of fold angle of this hinge must not cause Q_1 and Q_2 to collide. For the remainder of the paper, we assume that this range is nonempty.

The edges e^{Q_1} and e^{Q_2} can be mapped to edges on the unfoldings (P_1, F_1) and (P_2, F_2). Because we allow multiple

coverage of faces and edges in a folded state, it is possible for multiple edges in (P_1, F_1) to correspond to e^{Q_1}. Let E^{P_1} be the set of these edges, and similarly for E^{P_2}. By definition, if we are able to guarantee for a (P_3, F_3) that one edge in E^{P_1} will coincide with one edge in E^{P_2} in the folded state, then (P_3, F_3) will satisfy condition (1) of Problem 1. We therefore modify the problem statement slightly to concern folded states rather than their images.

Problem 2. *Given two unfoldings (P_1, F_1) and (P_2, F_2), an edge e^{P_1} in (P_1, F_1), and an edge e^{P_2} in (P_2, F_2), find an unfolding (P_3, F_3) such that*

1. *(P_3, F_3) is an unfolding of the union of translated and rotated instances of Q_1 and Q_2, the images of folded states of (P_1, F_1) and (P_2, F_2) respectively,*
2. *in the folded state, e^{P_1} and e^{P_2} coincide, and*
3. *(P_3, F_3) contains rotated, translated, and/or reflected instances of (P_1, F_1) and (P_2, F_2) as subsets.*

Solving this problem for any combination of $e^{P_1} \in E^{P_1}$ and $e^{P_2} \in E^{P_2}$ will satisfy Problem 1. The remainder of this paper is concerned with solving Problem 2.

2.1 Representation

For analysis of our algorithms, we assume a random-access machine that can perform real arithmetic and store real numbers. We store an unfolding (P, F) on this machine in four parts:

1. an $N \times 2$ array $\mathcal{V}_{(P,F)}$, where N is the total number of unique vertices in P and F. Row i of $\mathcal{V}_{(P,F)}$ contains the (x, y) coordinate values of the ith vertex.
2. an $M \times 2$ array $\mathcal{E}_{(P,F)}$, where M is the total number of edges in P and F. Row i of $\mathcal{E}_{(P,F)}$ contains the indices from $\mathcal{V}_{(P,F)}$ of the endpoints of edge i.
3. a list $\mathcal{B}_{(P,F)}$ of length M. Element i is an indicator equal to 1 if edge i is an edge on P and 0 if edge i is a fold line in F.
4. a list $\Theta_{(P,F)}$ of length M. Element i is the fold angle corresponding to edge i if edge i is a fold line and 0 otherwise.

Thus storage of (P, F) is $O(N+M)$. For the remainder of this paper, we will use the notation that N_i is the number of vertices in (P_i, F_i) and M_i the number of edges.

3 Edges on the Convex Hull Boundary

In order to construct (P_3, F_3), the input unfoldings (P_1, F_1) and (P_2, F_2) must be arranged in the plane without intersection and extra material added so that the edges to be joined, e^{P_1} and e^{P_2}, are coincident in the folded state. We use the following insight.

Lemma 1. *If e^{P_1} and e^{P_2} are on the boundaries of the convex hulls of their respective unfoldings, then they may be placed coincident in the plane and will not cause (P_1, F_1) and (P_2, F_2) to intersect.*

Proof. Let $\mathrm{CH}(P_1)$ be the convex hull of P_1. By definition, $P_1 \subseteq \mathrm{CH}(P_1)$. Because $\mathrm{CH}(P_1)$ is convex and e^{P_1} is an edge on

its boundary, $\mathrm{CH}(P_1)$ must lie entirely on one side of the line common to e^{P_1} (see Fig. 4). Similarly, $\mathrm{CH}(P_2)$ must lie entirely on one side of the line common to e^{P_2}.

Rotate, translate, and reflect (P_2, F_2) so that e^{P_2} is coincident to e^{P_1} and $\mathrm{CH}(P_2)$ is on the opposite side of e^{P_2} as $\mathrm{CH}(P_1)$. Since $\mathrm{CH}(P_1)$ and $\mathrm{CH}(P_2)$ are on opposite sides of the line now collinear to both e^{P_1} and e^{P_2}, they cannot intersect. Likewise, P_1 and P_2 cannot intersect. □

figure 4

In this case, the unfolding (P_3, F_3) is simply the union of (P_1, F_1) and the transformed (P_2, F_2), with the edge e^{P_1} (also e^{P_2}) converted from a boundary edge of P_1 (resp., P_2) to a fold in F_3.

When e^{P_1} or e^{P_2} is not on the boundary of the convex hull of its unfolding, then naïvely following the above procedure may lead to self-intersection of (P_3, F_3). However, it is possible to modify (P_1, F_1) and (P_2, F_2) without changing their corresponding folded structures so that the above procedure may be used. Taking the case of (P_1, F_1), this modification would consist of constructing an additional unfolding (P^b, F^b) and attaching it to (P_1, F_1) so that in the folded state e^{P_1} becomes coincident to an edge on the boundary of the combined unfolding's convex hull. We call (P^b, F^b) a *bridge* and say that e^{P_1} has been *bridged* to the boundary of the convex hull. As we will show in Section 4,

Lemma 2. *Given an unfolding (P, F) and an edge e^P, it is always possible to bridge e^P to the boundary of the convex hull. The bridge can be constructed in $O\left(M^2 + (N^2 + NM)\log(N + M)\right)$ time.*

Combining Lemmas 1 and 2 yields our main result.

Theorem 1. *For any unfoldings (P_1, F_1) and (P_2, F_2), and edges e^{P_1} in (P_1, F_1) and e^{P_2} in (P_2, F_2), there exists an unfolding (P_3, F_3) that satisfies Problem 2. Furthermore, this unfolding can be computed in time polynomial in the number of vertices and edges in the original unfoldings.*

Proof. According to Lemma 2, edges e^{P_1} and e^{P_2} can always be bridged to the boundaries of the convex hulls of P_1 and P_2 respectively. Let $e^{P_1}_{new}$ be the bridge edge on the boundary of the convex hull that coincides with e^{P_1} in the folded state, and similarly with $e^{P_2}_{new}$. Applying Lemma 1 to the modified (P_1, F_1) and (P_2, F_2) using $e^{P_1}_{new}$ and $e^{P_2}_{new}$ as the edges to join yields an unfolding (P_3, F_3) that satisfies the conditions of Problem 2.

Complexity: According to Lemma 2, bridging takes $O\left(M_i^2 + (N_i^2 + N_i M_i)\log(N_i + M_i)\right)$ time for each unfolding. Rotating and translating the modified unfoldings so that the new edges to join are coincident takes $O(\min(N_1, N_2))$ time (we only have to transform one unfolding). Thus the entire composition algorithm has complexity

$$O\left(M_1^2 + M_2^2 + (N_1^2 + N_1 M_1)\log(N_1 + M_1) + (N_2^2 + N_2 M_2)\log(N_2 + M_2)\right)$$

□

4 Constructing the Bridge

This section describes the algorithms and analysis that establish Lemma 2. There are two cases to consider:

1. e^P is on the boundary of P
2. e^P is a fold line in F

We show that in either case, it is possible to construct a bridge such that in the folded state, an edge on the boundary of the convex hull of the modified unfolding collapses onto e^P.

In the course of the discussion, we make frequent use of *paths*, which we restrict to be simple polygonal chains. In so doing, it becomes possible to represent a path p of length n as a finite list of the vertices $p = p_1 p_2 \dots p_n$ in the order they appear in the chain.

Case 1: Edges on the Boundary

The procedure when e^P is a edge on the boundary of P is based on the following lemma.

Lemma 3. *If e^P is on the boundary (i.e., $e^P \subset \partial P$), then there exists a path through the free space beginning at a point on e^P and ending on the boundary of the convex hull of P.*

Proof. Because P is a polygon, its convex hull $\mathrm{CH}(P)$ is also a polygon. The vertices of $\mathrm{CH}(P)$ are a subset of the vertices of P. If the vertices on the boundary $V(P) = \{v_1^P, v_2^P, \dots, v_n^P\}$ are numbered in clockwise direction, $\mathrm{CH}(P)$ can be represented as an increasing sequence $(i_1, i_2, \dots, i_m)$ such that $\{v_{i_1}^P, v_{i_2}^P, \dots, v_{i_m}^P\}$ are the m vertices of $\mathrm{CH}(P)$ in clockwise order. If $e^P = (v_j^P, v_{j+1}^P)$, let P_k^{CH} be a polygon bounded by the path $p_k = v_{i_k}^P v_{i_k+1}^P \dots v_{i_{k+1}}^P v_{i_k}^P$ where $i_k \leq j < i_{k+1}$. Since e^P is an edge of P, it is a boundary edge of P_k^{CH}. The convex hull edge $(v_{i_k}^P, v_{i_{k+1}}^P)$ is also on the boundary of P_k^{CH}. Finally, P_k^{CH} is not self-intersecting, else P would be self-intersecting or the convex hull would not completely contain P. Given these characteristics, a path from e^P to $\mathrm{CH}(P)$ that does not intersect with P or $\mathrm{CH}(P)$ except at the terminal vertices must exist inside P_k^{CH}. □

In order to bridge e^P to the boundary of the convex hull, we propose to compute such a path and overlay it with accordion-style pleats (an unfolding consisting of a sequence of non-intersecting folds with fold angles alternating between π and $-\pi$) so that the edge at the end of the path (the convex hull edge) collapses exactly onto e^P in the folded state. This is always possible by virtue of the following lemma.

Lemma 4. *Given a starting edge e^P and any simple path $p = p_1 p_2 \dots p_{n_e}$ such that the first vertex p_1 is the midpoint of e^P, there exists a series of pleats such that every p_i lies on a fold f_i, and in the folded state, all f_i are coincident to e^P.*

Proof. We prove the existence of such a pleat structure by construction. The pleat structure we produce is based on an isosceles trapezoid. If an isosceles trapezoid is folded with a fold angle of $\pm\pi$ down its axis of symmetry, then its two legs will coincide in the folded state. In a chain of isosceles trapezoids, where every trapezoid shares only its legs with its neighbors, folding every trapezoid down its axis symmetry will cause the legs of all the trapezoids to coincide. Therefore, a chain of folded isosceles trapezoids where every path vertex p_i lies on a leg of a trapezoid and e^P is also the leg of a trapezoid

is a pleat structure that satisfies the conditions of this lemma. The full algorithm for constructing this chain can be found in Alg. 1 and is illustrated in Fig. 5.

figure 5

algorithm 1

Using the perpendicular bisectors of the segments in p (line 3) ensures that the median of every trapezoidal pleat is exactly one segment of p, and that each newly created edge has as its midpoint the next vertex of p. The resulting pleats have a width of at most $\|e^P\|$. Line 9 makes the pleats non-self-intersecting. During the pleat construction in lines 2-8, three types of self-intersection can occur (see Fig. 5(c)).

1. When an edge to reflect e_i intersects with the perpendicular bisector, the resulting pleat must be trimmed to a triangle. Note that the triangle will still contain the path vertex.
2. When a pleat overlaps with the subsequent segment of p, it results in an intersection between adjacent pleats. Let e_i be the edge shared between the two intersecting pleats, and p_i be its midpoint. The two pleats are both trimmed to non-isosceles trapezoids that meet at p_i. This operation alone would cut the bridge into two pieces. Therefore, a second set of right triangular pleats must be added in the free space next to e_i to maintain connectivity. The addition of the triangular pleats causes a change in orientation of the pleats. If such a change in undesirable, the fold down the middle of the triangles can also be omitted.
3. When nonadjacent segments of p are close together, their corresponding pleats may overlap. The overlap may be resolved by dividing the overlapping region along the bisectors of the two conflicting segments of p. Since the path p is simple, the pleats will remain connected and the fold lines will still contain the path vertices.

In all cases, the modifications do not disconnect the pleats, so the pleats will be a valid unfolding. In addition, the fold lines remain in the same locations, so the pleat structure will satisfy the conditions of the lemma. □

It is also of interest to check the storage size required by the resulting pleat structure and the complexity of Alg. 1.

Lemma 5. *Algorithm 1 outputs an unfolding with $O(n_e)$ vertices and $O(n_e)$ edges.*

Proof. Each of the n_e iterations of the for loop in lines 2–8 adds a constant number of vertices and edges to the pleat structure. In line 9, type 1 intersections cause removal of two vertices and one edge for each intersection, of which there can be at most n_e. Type 2 intersections cause addition of two vertices and four edges each, and again there can be at most n_e. In type 3 intersections, the number of vertices added to the pleats is equal to twice the number of vertices involved in the intersection, plus four. We say that a vertex is involved in an intersection if it lies on the boundary of the intersecting region. It is clear that a vertex can only ever participate in at most one type 3 intersection. Therefore, the total number of vertices added to correct type 3 intersections is at most $O(n_e)$. Since the edges only form chains to connect the new vertices, the number of edges added is also $O(n_e)$. Thus the output unfolding of Alg. 1 has $O(n_e)$ vertices and $O(n_e)$ edges. □

Lemma 6. *Algorithm 1 takes $O((n_e+n_i)\log n_e)$ time, where n_i is the number of self-intersections detected in line 9.*

Proof. The for loop in lines 2–8 is executed $n_e - 1$ times and contains a body that is $O(1)$, so it takes $O(n_e)$ time total. In line 9, the pleats must be checked for each type of intersection. Type 1 and 2 intersections can only occur between neighboring pleats. They can be found and corrected simply by iterating through the pleats in order, an operation that takes $O(n_e)$ time. Detecting type 3 intersections, on the other hand, requires a check for self-intersection of P^b. Naïvely checking every pair of edges on the boundary of P^b for intersection would take $O(n_e^2)$. It is usually faster to use the Bentley-Ottman sweep line algorithm [22], which is $O((n_e + n_i)\log n_e)$ time, where n_i is the number of intersections.

Summing over the entire algorithm yields an overall complexity of $O((n_e + n_i)\log n_e)$. Note that in the worst case, when n_i is $O(n_e^2)$, the complexity can be reduced to $O(n_e^2)$ if the naïve collision checking method is used. □

We now give the full bridge constructing algorithm (Alg. 2), illustrated in Fig. 6.

figure 6

algorithm 2

Line 1: Find a Path to the Convex Hull Boundary.

According to Lemma 3, a path from e^P to the boundary of the convex hull must exist. Theoretically, any such path will suffice. Since the goal is to overlay the path with pleats, we choose a path surrounded on both sides by as much free space as possible. A path along the straight skeleton of P_k^{CH} satisfies this criterion.

The straight skeleton of P_k^{CH} is the union of trajectories of vertices of P_k^{CH} when the boundary of P_k^{CH} is shrunk in such a way that all edges retain their orientation and move towards the interior of P_k^{CH} at the same speed. The straight skeleton was introduced in [23] as an alternative to the medial axis, which can contain parabolic arcs when P_k^{CH} is nonconvex. As its name suggests, the straight skeleton contains only line segments. When p_k is the boundary of an n_e-gon, the straight skeleton is a tree with $(n_e - 2)$ vertices and $(2n_e - 3)$ edges that partitions the polygon into n_e regions, each of which contains one segment of p_k [23]. It can be computed straightforwardly by simulating the polygon shrinking process in $O(n_e^2 \log(n_e))$ time, although by using more complex data structures, a complexity of $O(n_e^{17/11+\varepsilon})$ can be achieved [24].

The path that we use to construct the pleats is the path in the straight skeleton from the region containing e^P to the region containing the convex hull edge (Fig. 6(c)). Since the straight skeleton is a tree, there will only be one such path. Let $p = p_1 p_2 \ldots p_{n_e}$ be this path , with the first vertex p_1 located at the midpoint of e^P, intermediate vertices $p_2, \ldots, p_{n_e-1}$ at vertices in the straight skeleton, and the final vertex p_{n_e} on the convex hull edge.

Line 2–3: Overlay Pleats.

Overlay p with accordion-style pleats using Alg. 1. For the last pleat, rather than following the procedure in lines 3–5 of Alg. 1, the point of intersection between the line common to edge e_{n_e-1} and the line common to the convex hull edge is found. Then, e_{n_e-1} is rotated about this point of intersection onto the convex hull edge to create e_{n_e}. This guarantees that the last edge added to the pleated structure lies on the boundary of the unfolding's convex hull and that the new pleat is still an isosceles trapezoid. As in Alg. 1, e_{n_e} is then assigned a fold angle of π, and a fold line is added on the new pleat's axis of symmetry with a fold angle of $-\pi$. The last edge's exact location on the convex hull boundary is not prespecified. Since,

however, p_{n_e-1} is a vertex on the straight skeleton bordering the region containing the convex hull edge, the median of this pleat will lie entirely inside the free space. The result of this step is a bridge that collapses flat onto the face adjacent to e^P.

Line 4: Remove Intersections with the Input Unfolding.

Although the path p lies entirely in the free space, the pleats following p have a width and may intersect with the input unfolding (Fig. 7). In this case, the overlapping regions can be cut out of the bridge. When this operation causes the bridge to become disconnected, then pieces that are not connected to e^P should also be removed. The bridge will still extend from e^P to the convex hull boundary since p lies entirely in the free space.

figure 7

Finally, if a pleat is so long that it interferes with the folding of P, it can be trimmed or fold lines can be added to crimp the pleat arbitrarily small (Fig. 8).

figure 8

Lemma 7. *Algorithm 2 outputs an unfolding with $O(N)$ vertices and $O(M+N)$ edges.*

Proof. According to Lemma 5, since the length of the path p is $O(N)$, the bridge will have $O(N)$ vertices and $O(N)$ edges. Merging the bridge into the unfolding in line 3 decreases the number of total vertices by two and the number of edges by one. Finally, removing intersections and trimming pleats in line 4 can add at most N vertices and M edges to the unfolding. Therefore, the output unfolding must have $O(N)$ vertices and $O(M+N)$ edges. □

Lemma 8. *Algorithm 2 computes a bridge in $O\left(N^2+NM\log(N+M)\right)$ time.*

Proof. Line 1 of the algorithm finds a path to the boundary of the convex hull. This requires first computing the convex hull $\mathrm{CH}(P)$, which can be done in $O(N)$ time [25, 26]. The region P_k^{CH} can be found by following the boundary of P starting at e^P until a vertex that is also a vertex of the convex hull is reached. Tracing this path will also take $O(N)$ time and will result in a region P_k^{CH} bordered by a path of length $O(N)$. The straight skeleton of P_k^{CH} will thus contain $O(N)$ vertices and $O(N)$ edges [23], and it can be computed in $O(N^{17/11+\varepsilon})$ time [24]. Finally, the path p is found via a search over the straight skeleton tree that will, in the worst case, require traversal over all the edges in the tree, taking $O(N)$ time. The length of path p is also $O(N)$.

Line 2 calls Alg. 1, which takes $O(N^2)$ time for a path of length $O(N)$ and produces a pleat structure with $O(N)$ vertices and $O(N)$ edges. Line 4 requires another check for intersections, this time between the boundary of the pleats and the the boundary of P. Since the only intersections will occur between the pleats and the unfolding (P,F), the number of intersections is $O(NM)$, and line 4 will take at worst $O((N+M+NM)\log(N+M)) = O(NM\log(N+M))$ time [22]. Lines 3 and 5 are $O(1)$ operations.

Summing over all the lines in the algorithm yields an overall complexity of $O(N^2+NM\log(N+M))$ for Alg. 2. □

Case 2: Edges that are Fold Lines

When e^P is not on the boundary of P, then it is not adjacent to any free space and Alg. 2 alone cannot be used. Instead, we must first construct a boundary edge e^P_{ref} that in the folded state will coincide with e^P. This can be achieved by taking a path through the interior of P to an edge e_b on the boundary and reflecting the path out of P. Algorithm 3, illustrated in Fig. 9, gives the procedure for constructing a bridge for this case.

figure 9

algorithm 3

This case requires construction of the *edge-adjacency graph* (Fig. 10) of the unfolding (P,F). This is a graph $G^e = (\mathcal{V}^e, \mathcal{E}^e)$ with vertices in $\mathcal{V}^e$ located at the midpoints of the edges in P and F, and edges $\mathcal{E}^e = \{(v^e_i, v^e_j) | e_i$ and e_j lie on the boundary of the same face in $(P,F)\}$. The edge-adjancency graph has the following properties:

- Every edge in $\mathcal{E}^e$ corresponds to a single face in (P,F)
- A path in G^e corresponds to a connected set of faces in (P,F)
- G^e has M vertices and $O(M^2)$ edges
- G^e can be constructed in $O(M^2)$ time [27]

figure 10

Line 1: Choose A Boundary Edge.

Before constructing the bridge, it is necessary to choose where to attach it to the unfolding. Since e^P is not on the boundary of P, it is impossible to connect the bridge at e^P without causing self-intersection of the final unfolding. Any boundary edge e_b can be used here. For our implementation, we chose e_b to minimize the length of the final constructed bridge. We estimated the length of the bridge using the sum of the lengths of 1) the path from e^P to e_b in the edge-adjacency graph G^e and 2) the path from e_b to the boundary of the convex hull in the straight skeleton produced when calling Alg. 2. Note that to calculate this cost, we generate the straight skeleton of every region of free space P^{CH}_k in the unfolding's convex hull. There are $O(N)$ such regions.

Line 2: Bridge e_b to the Convex Hull Boundary.

Unless e_b is on the boundary of P's convex hull, the faces between e^P and e_b cannot simply be reflected over e_b to make e^P a boundary edge, since this operation may result in intersection with P. Instead, the edge e_b must first be bridged to the boundary of the convex hull using Alg. 2. Let e_{CH} be the resulting edge on the convex hull boundary.

Line 3: Reflect Faces between e^P and e_{CH}.

In order to construct a e^P_{ref} on the boundary of the unfolding, reflect all faces between e^P and e_{CH} over e_{CH}. These faces can be found by consulting the edge-adjacency graph G^e. A path in G^e from the vertex corresponding to e^P to the vertex corresponding to e_{CH} yields a set of faces that connect e^P and e_{CH}. Since e_{CH} is on the convex hull boundary, the reflected faces will not intersect with the rest of the unfolding.

Line 4: Bridge e^P_{ref} to the Convex Hull Boundary.

The result of line 3 is that e^P_{ref} is now on the boundary of P but not necessarily of the convex hull, reducing the situation to Case 1. In the folded state, the reflected path will fold flat along the surface of the input folded structure Q so that the reflected edge is coincident e^P. Thus any material attached to e^P_{ref} is as if it were added at e^P.

Lemma 9. *Algorithm 3 outputs an unfolding with $O(N)$ vertices and $O(M+N)$ edges.*

Proof. According to Lemma 7, line 2 will increase the size of the unfolding to have $O(N)$ vertices and $O(M+N)$ edges. Since line 3 reflects only faces that already exist, it can at most double the size of the unfolding. A final application of Alg. 2 in line 4 yields an output unfolding with $O(N)$ vertices and $O(M+N)$ edges. □

Lemma 10. *Algorithm 3 computes a bridge in $O\left(M^2+(N^2+NM)\log(N+M)\right)$ time.*

Proof. Line 1 of the algorithm chooses a boundary edge e_b. This requires constructing the edge-adjacency graph of the unfolding in $O(M^2)$ time, as well as straight skeletons in $O(N^{17/11+\varepsilon})$ time. The closest boundary edge to e^P can be found on the resulting $O(M+N)$-vertex, $O(M^2+N)$-edge graph in $O((M+N)^2)$ time using Dijkstra's algorithm [28].

Line 2 calls Alg. 2, which by Lemmas 7 and 8 produces an unfolding with $O(N)$ vertices and $O(N+M)$ edges in $O\left(N^2+NM\log(N+M)\right)$ time. Since the path from e^P to e_b was already calculated in line 1, line 3 incurs only the cost of reflecting $O(N+M)$ faces in $O(N+M)$ time. Finally, line 4 takes $O\left(N^2+(N^2+NM)\log(N+M)\right)=O\left((N^2+NM)\log(N+M)\right)$, according to Lemma 8.

Summing over all the lines in the algorithm yields an overall complexity of $O\left(M^2+(N^2+NM)\log(N+M)\right)$ for Alg. 3.

□

4.1 Summary

We have now provided enough detail to support Lemma 2:

Lemma 2. *Given an unfolding (P,F) and an edge e^P, it is always possible to bridge e^P to the boundary of the convex hull. The bridge can be constructed in $O\left(M^2+(N^2+NM)\log(N+M)\right)$ time.*

Proof. The previous sections detail how to construct a bridge when e^P is a boundary edge or e^P is a fold line. The edge to join e^P must fall into one of these cases. It is therefore always possible to bridge e^P to the convex hull.

Complexity: According to Lemma 8, constructing a bridge when e^P is a boundary edge takes $O\left(N^2+NM\log(N+M)\right)$ time. Lemma 10 states that constructing a bridge when e^P is a fold line takes $O\left(M^2+(N^2+NM)\log(N+M)\right)$ time. Which of these two cases is applicable can be determined by consulting the list $\mathcal{B}_{(P,F)}$ in $O(1)$ time. Thus, the overall complexity of constructing a bridge for arbitrary e^P is $O\left(M^2+(N^2+NM)\log(N+M)\right)$. □

5 Experimental Results

figure 11

The proposed algorithm was implemented in MATLAB and tested on various compositions. Figure 11 shows the input unfoldings and the composition for each test. Cut lines on the boundary of the unfoldings are shown as solid, and folds are shown in dotted lines. The edges to join are shown in bold on the input unfoldings. The expected folded states of each generated unfolding are simulated and verified via physical models folded from poster board.

All final folded states are, as expected, the two inputted surfaces connected along the specified edge. Since the joined edges act as a hinge joint, the angle of the input surfaces relative to each other in the folded state are not fixed. This effect can be clearly seen in the physical models, where the stiffness of the material used prevented faces from resting coincident as they do in the simulated folded states. The constructed bridges (shaded) are indeed a series of pleats connecting the edges to join to the boundaries of the convex hulls of their respective unfoldings. The fourth row of Fig. 11 shows the compositions with only the bridges folded. The edges to join coincide with each other in the folded state.

Figure 11(a) joining two cubes is an example of Alg. 2. Both edges to join are on the boundaries of their unfoldings. Pleats are added to the unfolding on the right since the edge to join is not already on the boundary of the convex hull. In the folded state, these pleats are flattened between the two cubes.

Figures 11(b)–(d) demonstrate Alg. 3, when an edge to join is on the interior of the unfolding. The square pyramid in Fig. 11(b) is the same one considered in Fig. 9. Not only are pleats added but a face of the unfolding is reflected in the bridge construction. In the folded state, this face lies flat against the surface of the square pyramid so that one side is doubly covered. Similarly, for the insect and gripper in Fig. 11(d), faces between the edges to join and the boundaries of their unfoldings are reflected to create the bridge. Since the boundary edges where the bridges are attached are already on the convex hulls of the unfoldings, no extra pleats are added.

6 Discussion and Future Work

This paper demonstrates that a connected composite unfolding satisfying the requirements of Problem 2 exists. Although an unfolding could be found in every case, however, it was not necessarily the most efficient unfolding in terms of material usage. For example, Fig. 11(a) shows a composition of two cubes that could be achieved simply by joining the two input unfoldings along the chosen edges. The unfolding produced by our algorithm uses extra pleats since it seeks to join unfoldings at their convex hulls.

In addition, practically, factors other than connectivity between the two input unfoldings must be taken into account. For example, when choosing the boundary edge in Alg. 3, we minimized the amount of extra material added. Other times, it may be more desirable to minimize the number of layers or to maximize the width of pleats. Each of these optimization problems yields a different choice when performing Alg. 2 or 3. For example, even if e^P is on the boundary, taking a longer path through the interior of the unfolding to an area with a greater amount of free space may yield wider pleats or fewer layers, so we may want to use Alg. 3 even if Alg. 2 is applicable. This was the approach used for the test in Fig. 11(c).

This work considers connecting two unfoldings along an edge, which is the minimum attachment necessary for two surfaces to be joined. This leads to a folded state where the two input surfaces can move relative to each other. Another type of joining is a face-joining, where the two input surfaces would be attached along one or multiple faces and would

fixed relative to each other. An extension of the proposed algorithm to face-joinings is straightforward: repeat the algorithm for every boundary edge of the two connecting surfaces, choose the result that minimizes the amount of added material (or optimizes some other metric), and fix the hinge fold angles so that the two surfaces coincide. Of course, while theoretically the folded state would in this case be constrained so that the joined surfaces touch, a physical instantiation would be no stiffer than that produced when seeking a hinge joint alone. We are currently investigating alternative methods for composing unfoldings when joining occur along faces.

Finally, the results of this algorithm are restricted in that the generated unfolding must contain (P_1, F_1) and (P_2, F_2) in their entirety. When humans compose origami designs, they often rearrange the unfoldings and change the shape of the free space to achieve more efficient composite unfoldings (see, for example, Fig. 1). As discussed in Section 1, depending on the application, certain folds should not be cut; however, a 3-D surface often has several equivalent unfoldings that yield the same mechanical strength. In order for this algorithm to be useful practically, we will analyze the mechanical properties of materials that have been folded as compared to cut and develop a model that will allow us to generate equivalent unfoldings.

7 Conclusion

This paper addresses automatic composition of unfoldings of 3-D surfaces. We show that given the unfoldings of two 3-D surfaces, it is always possible to construct a bridge between them such that the folded state is the two originals connected along a hinge joint, and we provide a polynomial-time algorithm to generate the composite unfolding. The algorithm was tested on a variety of simple compositions, demonstrating that it can indeed be used to generate one-piece unfoldings of composed surfaces. The algorithm shows promise for automated design of folded structures in the future.

Acknowledgment

We thank Cagdas Onal for helpful discussions. This work was funded in part by NSF grants 1240383 and 1138967. C.S. was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

References

[1] Sung, C., Demaine, E. D., Demaine, M. L., and Rus, D., 2013. "Joining unfoldings of 3-D surfaces". In Proc. ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference(IDETC/CIE), no. DETC2013-12692.

[2] Onal, C. D., Wood, R. J., and Rus, D., 2013. "An origami-inspired approach to worm robots". *IEEE/ASME Transactions on Mechatronics,* ***18***(2), pp. 430–438.

[3] Theis, H. E., 1999. *Handbook of Metalforming Processes*. CRC Press, New York, NY.

[4] Okuzaki, H., Saido, T., Suzuki, H., Hara, Y., and Yan, H., 2008. "A biomorphic origami actuator fabricated by folding a conducting paper". *Journal of Physics: Conference Series,* ***127***(1), p. 012001.

[5] Siegel, A. C., Phillips, S. T., Dickey, M. D., Lu, N., Suo, Z., and Whitesides, G. M., 2009. "Foldable printed circuit boards on paper substrates". *Advanced Functional Materials,* ***20***(1), pp. 28–35.

[6] Hawkes, E., An, B., Benbernou, N. M., Tanaka, H., Kim, S., Demaine, E. D., Rus, D., and Wood, R. J., 2010. "Programmable matter by folding". *Proceedings of the National Academy of Sciences,* ***107***(28), pp. 12441–12445.

[7] Hoover, A. M., Steltz, E., and Fearing, R. S., 2008. "RoACH: An autonomous 2.4g crawling hexapod robot". In Proc. 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 26–33.

[8] Felton, S. M., Tolley, M. T., Onal, C. D., Rus, D., and Wood, R. J., 2013. "Towards autonomous self-folding: A printed inchworm robot". In Proc. 2013 IEEE International Conference on Robotics and Automation (ICRA).

[9] Whitney, J. P., Sreetharan, P. S., Ma, K. Y., and Wood, R. J., 2011. "Pop-up book MEMS". *Journal of Micromechanics and Microengineering,* ***21***(11), p. 115021.

[10] Demaine, E. D., and O'Rourke, J., 2008. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press.

[11] Bern, M., Demaine, E. D., Eppstein, D., Kuo, E., Mantler, A., and Snoeyink, J., 2003. "Ununfoldable polyhedra with convex faces". *Computational Geometry,* ***24***(2), pp. 51–62.

[12] Tachi, T., 2006. "3D origami design based on tucking molecule". In Proc. 4th International Conference on Origami in Science, Mathematics, and Education (4OSME).

[13] Tachi, T., 2010. "Origamizing polyhedral surfaces". *IEEE Transactions on Visualization and Computer Graphics,* ***16***(2), pp. 298–311.

[14] Mitani, J., 2009. "A design method for 3d origami based on rotational sweep". *Computer-Aided Design and Applications,* ***6***(1), pp. 69–79.

[15] Mitani, J., 2011. "A design method for axisymmetric curved origami with triangular prism protrusions". In Proc. 5th International Conference on Origami in Science, Mathematics, and Education (5OSME).

[16] Cheng, H. Y., and Cheong, K. H., 2012. "Designing crease patterns for polyhedra by composing right frusta". *Computer-Aided Design,* ***44***(4), pp. 331–342.

[17] Bush, R., and Sèquin, C., 1999. "Synthesis of bent sheet metal parts from design features". In Proc. 5th ACM Symposium on Solid Modeling and Applications, pp. 119–129.

[18] Patel, J., and Campbell, M. I., 2008. "An approach to automate concept generation of sheet metal parts based on manufacturing operations". In Proc. ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE), no. DETC2008-49648.

[19] Patel, J., and Campbell, M. I., 2010. "An approach to automate and optimize concept generation of sheet metal parts by topological and parametric decoupling". *Journal of Mechanical Design,* ***132***, p. 051001.

[20] Wang, C.-H., 1997. "Manufacturability-driven decomposition of sheet metal products". PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[21] Currier, D. W., 1980. "Automation of sheet metal design and manufacturing". In Proc. 17th Conference on Design Automation, pp. 134–138.

[22] Bentley, J. L., and Ottmann, T. A., 1979. "Algorithms for reporting and counting geometric intersections". *IEEE Transactions on Computers,* ***100***(9), pp. 643–647.

[23] Aichholzer, O., Aurenhammer, F., Alberta, D., and Gärtner, B., 1995. "A novel type of skeleton for polygons". *Journal of Universal Computer Science,* ***1***(12), pp. 752–761.

[24] Eppstein, D., and Erickson, J., 1999. "Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions". *Discrete & Computational Geometry,* ***22***(4), pp. 569–592.

[25] McCallum, D., and Avis, D., 1979. "A linear algorithm for finding the convex hull of a simple polygon". *Information Processing Letters,* ***9***(5), pp. 201–206.

[26] Melkman, A. A., 1987. "On-line construction of the convex hull of a simple polyline". *Information Processing Letters,* ***25***(1), pp. 11–12.

[27] De Berg, M., Cheong, O., and Van Kreveld, M., 2008. *Computational Geometry: Algorithms and Applications*. Springer.

[28] Dijkstra, E. W., 1959. "A note on two problems in connexion with graphs". *Numerische Mathematik,* ***1***, pp. 269–271.

Figure Captions

1. *Left*: Walking and gripper robots folded out of the patterns shown. *Right*: The *composition*, a walking-gripping robot, whose unfolding was designed manually. Our goal is to generate such an unfolding automatically. *Credit: Robots were designed by Cagdas Onal and Michael Tolley.*
2. Q can be *unfolded* into P if Q is the image of a folded state $\phi_F(P)$
3. A fold f has a fold line and a fold angle
4. Two convex polygons placed next to each other are guaranteed not to intersect
5. Algorithm 1: Constructing pleats to follow a path. (a) The edge e^P and the path p to follow. (b) Reflected edges e_i (solid) and perpendicular bisectors $e_i^{\perp}$ (dotted). (c) Resulting pleats. Types 1, 2, and 3 self-intersections are shaded. (d) Pleats with self-intersections corrected. (e) Folded state of pleats. All e_i coincide.
6. Algorithm 2: Bridging an edge on the boundary of the unfolding to the boundary of the convex hull. (a) Original fold pattern. The edge to join e^P is bold and the convex hull is shown in gray. (b) The region P_k^{CH} (shaded) and its straight skeleton. (c) The path p from e^P to the boundary of the convex hull. (d) Pleats tiled along the path. (e) Output unfolding with the bridge added.
7. (a) A bridge (shaded) that intersects with (P,F). (b) The offending region is removed.
8. A long pleat. *Top*: The unfolding with offending pleat (shaded). *Bottom*: Folded state. (a) In the folded state, the pleat protrudes outside the adjacent face. (b) It can be crimped to not interfere with other folds and (c) trimmed to avoid protrusions.
9. Algorithm 3: Bridging an edge on the interior of the unfolding to the boundary of the convex hull. (a) Original fold pattern. The edge to join e^P is bold and the convex hull is shown in gray. (b) The edge-adjacency graph with the path from e^P to e_b in bold. (c) Pleats attached to e_b using Alg. 2. (d) The accordion path and interior faces reflected over the boundary of the convex hull. The convex hull is also updated. (e) Pleats attached to e_{ref}^P using Alg. 2. (f) Output unfolding with the bridge added.
10. Example edge-adjacency graph
11. Fold patterns generated by this algorithm. *Top*: Input fold patterns for the polyhedral complexes to join. The bold edges are the edges to join. *Second row*: The generated composite unfolding. Bridges constructed by our algorithm are shaded. *Third row*: The folded state of the composite unfolding. *Fourth row*: Physical model of the composition with just the bridge folded. *Bottom*: Physical models of the input surfaces and the composition folded from poster board.

Algorithm 1: CREATEPLEATS(e^P, p)

Data: $e^P = (v_1^P, v_2^P)$ = starting edge
$p = p_1 p_2 \dots p_{n_e}$ = path to follow
Result: unfolding (P^b, F^b) = pleats satisfying Lemma 4

// Beginning with e^P,
$v_{1,1} \leftarrow v_1^P$; $v_{1,2} \leftarrow v_2^P$;
// Compute fold line locations
for $i = 1, \dots, n_e - 1$ **do**
  $\ell_i^\perp \leftarrow$ perpendicular bisector of segment $p_i p_{i+1}$;
  $v_{i+1,1} \leftarrow$ reflection of $v_{i,1}$ over $\ell_i^\perp$;
  $v_{i+1,2} \leftarrow$ reflection of $v_{i,2}$ over $\ell_i^\perp$;
  $e_i^\perp \leftarrow \left(\frac{1}{2}(v_{i,1} + v_{i+1,1}), \frac{1}{2}(v_{i,2} + v_{i+1,2})\right)$;
  $e_{i+1} \leftarrow (v_{i+1,1}, v_{i+1,2})$;
end
Remove intersections;
// Unfolding of pleated structure
$P^b \leftarrow$ polygon bounded by the path $v_{1,1} v_{2,1} \dots v_{n_e,1} v_{n_e,2} \dots v_{2,2} v_{1,2}$;
$F^b \leftarrow \left\{(e_1^\perp, -\pi)\right\} \cup \bigcup_{i=2,\dots,n_e-1} \left\{(e_i, \pi), (e_i^\perp, -\pi)\right\}$;

Algorithm 2: BRIDGEFROMBOUNDARY$((P,F),e^P)$

Data: (P,F) = input unfolding
e^P = boundary edge to bridge
Result: (P_{new},F_{new}) = unfolding containing (P,F)
e^P_{new} = edge on CH(P_{new}) that folds onto e^P

1 $p \leftarrow$ path from e^P to CH(P) (Lemma 3);
2 $(P^b,F^b) \leftarrow$CREATEPLEATS(e^P,p);
3 $P_{new} \leftarrow P \cup P^b$; $F_{new} \leftarrow F \cup F^b \cup \{(e^P,\pi)\}$;
4 Remove intersections;
5 $e^P_{new} \leftarrow e_{n_e}$ created during bridge construction in line 2;

Algorithm 3: BRIDGEFROMFOLDLINE$((P,F),e^P)$

Data: (P,F) = input unfolding
e^P = fold line edge to bridge
Result: (P_{new},F_{new}) = unfolding containing (P,F)
e^P_{new} = edge on CH(P_{new}) that folds onto e^P

1 $e_b \leftarrow$ a boundary edge on P;
2 $\{(P_2,F_2),e_{\text{CH}}\} \leftarrow$
BRIDGEFROMBOUNDARY$((P,F),e_b)$;
3 $\{(P_3,F_3),e^P_{ref}\} \leftarrow$REFLECTFACES$((P_2,F_2),e^P,e_{\text{CH}})$;
4 $\{(P_{new},F_{new}),e^P_{new}\} \leftarrow$
BRIDGEFROMBOUNDARY$((P_3,F_3),e^P_{ref})$;

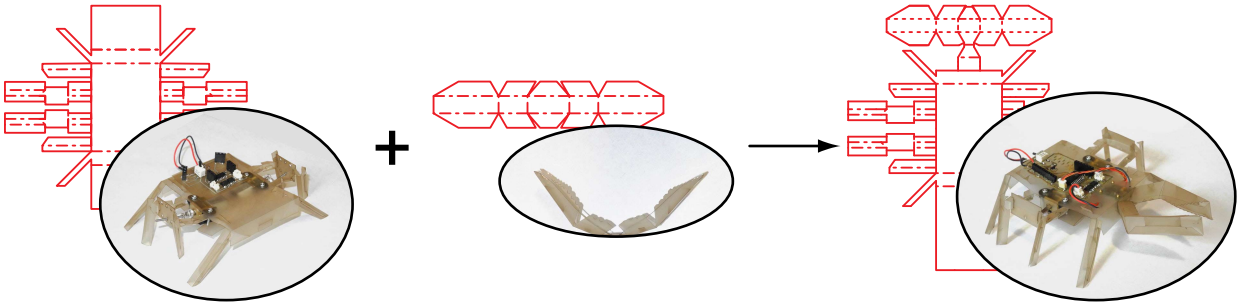


Fig. 1. fig1.eps

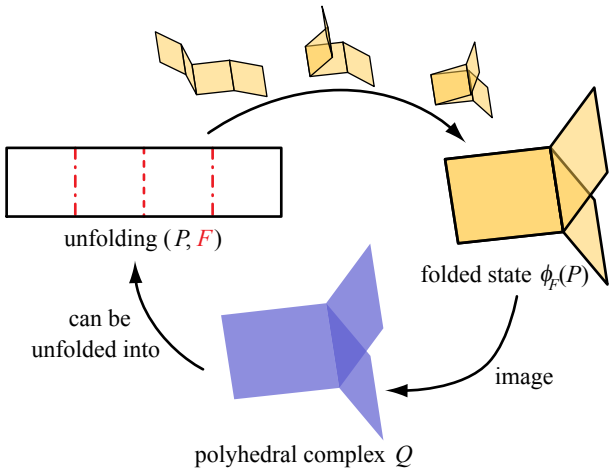


Fig. 2. fig2.eps

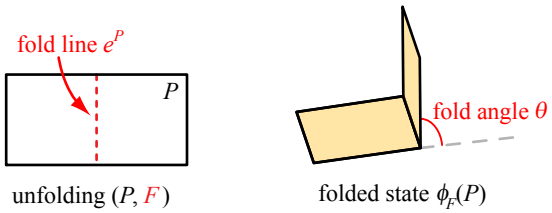


Fig. 3. fig3.eps

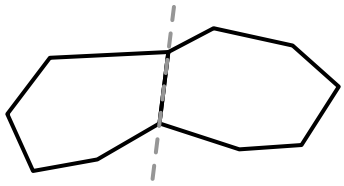
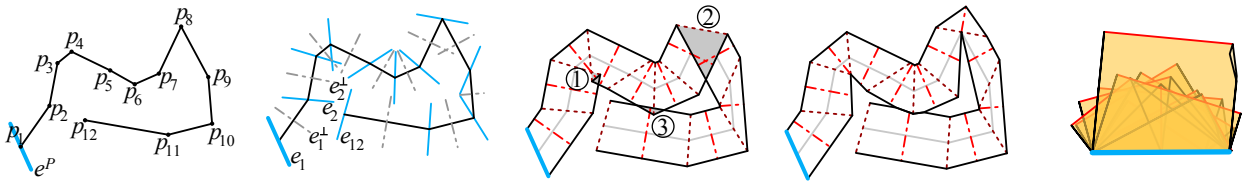
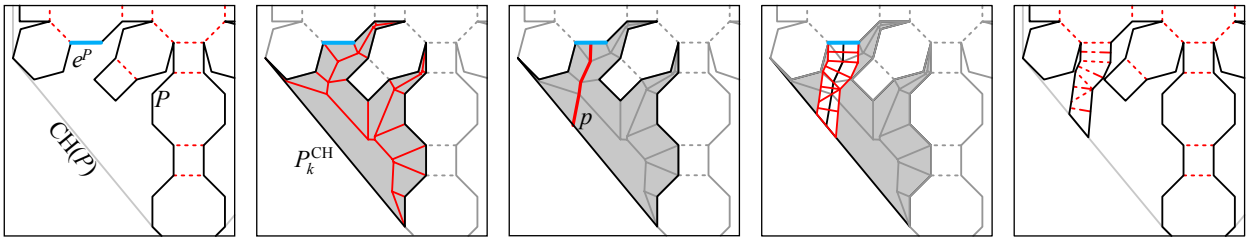


Fig. 4. fig4.eps



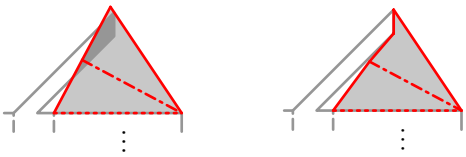
(a) Input (b) Lines 2-8 (c) Self-intersections (d) Line 9 (e) Folded state

Fig. 5. fig5.eps



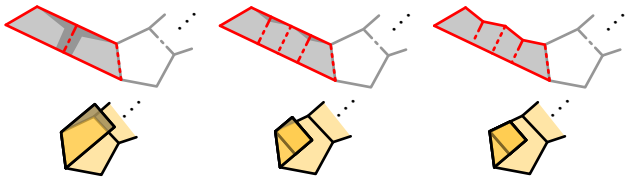
(a) Input (b) P_k^{CH} and straight skeleton (c) Line 1 (d) Line 2 (e) Output

Fig. 6. fig6.eps



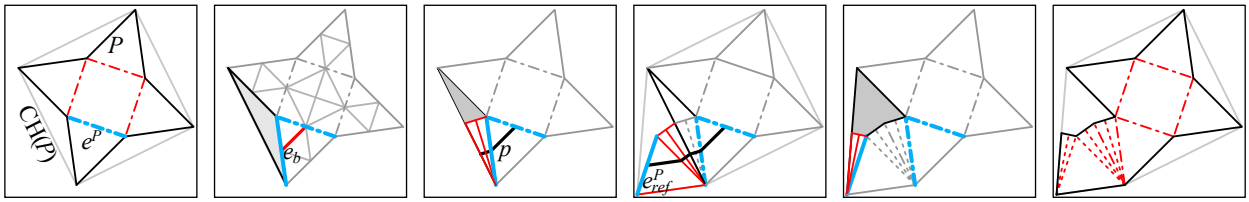
(a) Before (b) After

Fig. 7. fig7.eps



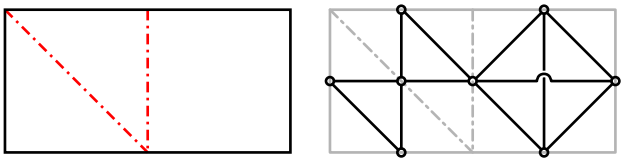
(a) Before (b) Crimped (c) Trimmed

Fig. 8. fig8.eps



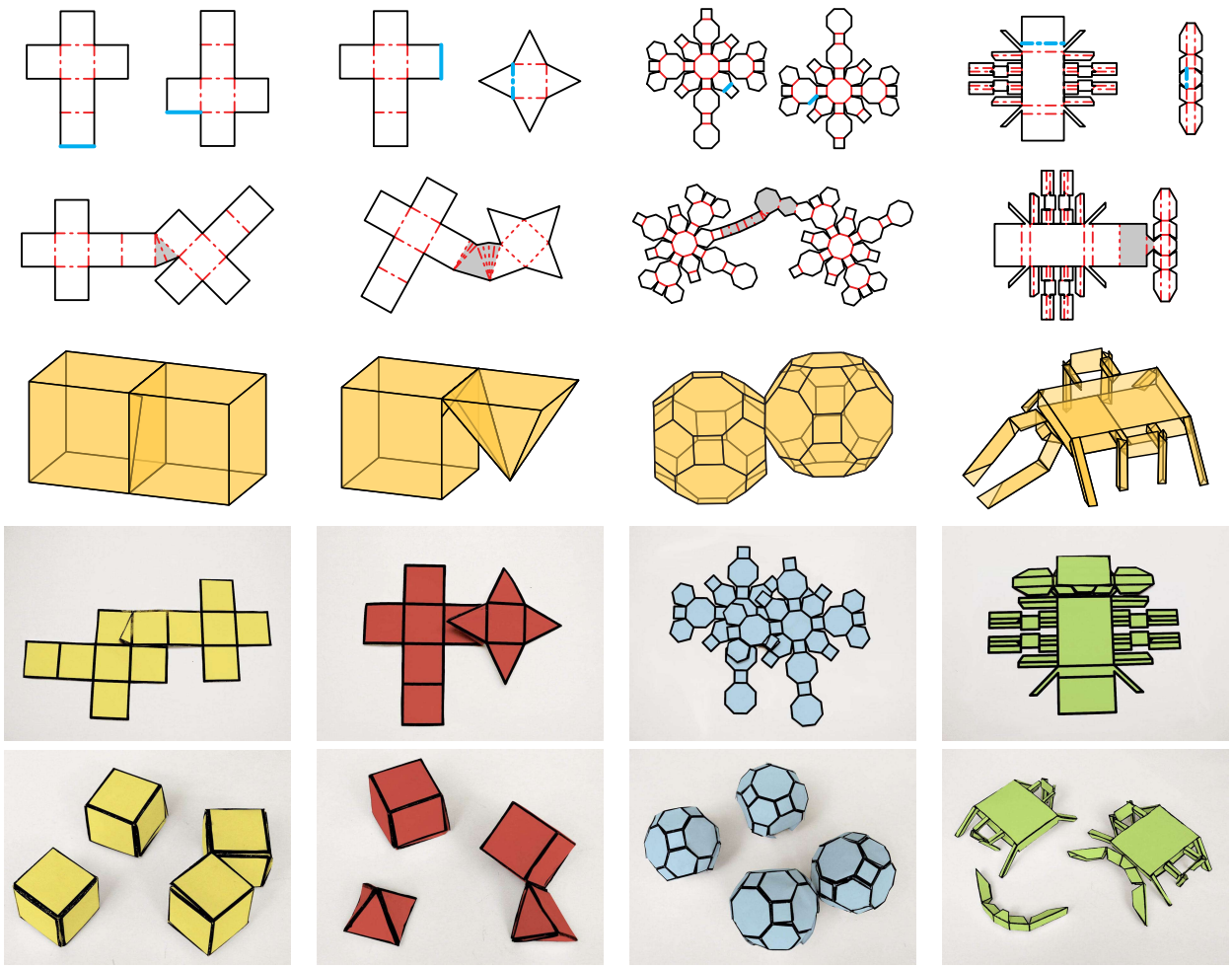
(a) Input (b) Line 1 (c) Line 2 (d) Line 3 (e) Line 4 (f) Output

Fig. 9. fig9.eps



(a) Unfolding (b) Edge-adjacency graph G^e

Fig. 10. fig10.eps



(a) Two cubes

(b) Cube and square pyramid

(c) Two truncated cuboctohedra

(d) Walking and gripping robots

Fig. 11. fig11.eps