1 Introduction

Today’s engineering designs are limited by practical considerations of their fabrication. Although recent advancements in machining practices and 3D 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 2D fabrication techniques with folding algorithms to allow rapid fabrication of 3D 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. 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 original. 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.

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.

1.1 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 Ref. [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, nonoverlapping 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 2D sheet of paper. Faces of the polyhedral surface are positioned on the 2D 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
Automate the design process. For example, [17] proposes a multi-state algorithm, allowing greater flexibility to reduce cost and 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.

1.2 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 Sec. 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 \( \partial P \) of \( P \), and the area enclosed is its interior \( P \). The line segments on the boundary are edges, denoted \( E(P) = \{ e_1, e_2, \ldots \} \), and the points where two edges meet are vertices, denoted \( V(P) = \{ v_1, v_2, \ldots \} \).

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 \( \phi_P : P \to \mathbb{R}^3 \) where the restriction map \( \phi_P : P \to \mathbb{R}^3 \) is isometric and non-crossing. We say that a polyhedral complex \( Q \) can be unfolded into \( P \) if \( Q \) is the image of a folded state \( \phi_P(P) \) (see Fig. 2). The folded state \( \phi_P(P) \) can also be represented as the union of polygonal faces. Every edge \( e^P \) of the folded state that is not on the boundary corresponds to a fold \( f = (e^P, \theta) \), where the line segment \( e^P \) is the line segment on \( P \) that corresponds to \( e^P \) and the fold angle \( \theta \) is equal to the dihedral angle between the faces of \( \phi_P(P) \) sharing \( e^P \) (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 dashed–dotted segments. We denote the set of folds $F = \{(e'_1, \theta_1), (e'_2, \theta_2), \ldots \}$. 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 $P \cup \bigcup F$, the free space $O$.

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

- translation by a vector $v$: all folds in $F$ are translated by $v$
- rotation by an angle $\alpha$ about a point $x$: $P$ and all edges in $F$ are rotated by $\alpha$ about $x$
- 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 $e_Q \in L(Q_1)$ and $e_Q \in L(Q_2)$, 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 $e_Q$ is coincident to $e_{Q'}$ 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 $e_{Q'}$ and $e_{Q''}$ 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 $e_{Q'}$ and $e_{Q''}$ 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 $e_{Q'}$. 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 $e_{Q'} \in (P_1,F_1)$, and an edge $e_{Q''} \in (P_2,F_2)$, 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 $(P_2,F_2)$, and
2. $(P_3,F_3)$ 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 $e_{Q'} \in E^{P_1}$ and $e_{Q''} \in E^{P_2}$ 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 $V_{(P,F)}$, where $N$ is the total number of unique vertices in $P$ and $F$. Row $i$ of $V_{(P,F)}$ contains the $(x, y)$ coordinate values of the $i$th vertex.
2. an $M \times 2$ array $E_{(P,F)}$, where $M$ is the total number of edges in $P$ and $F$. Row $i$ of $E_{(P,F)}$ contains the indices from $V_{(P,F)}$ of the endpoints of edge $i$.
3. a list $B_{(P,F)}$ 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 $T_{(P,F)}$ 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, $e_{Q'}$ and $e_{Q''}$, are coincident in the folded state. We use the following insight.

**LEMMA 1.** If $e_{Q'}$ and $e_{Q''}$ 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 $CH(P_i)$ be the convex hull of $P_i$. By definition, $P_i \subseteq CH(P_i)$. Because $CH(P_i)$ is convex and $e_{Q'}$ is an edge on its boundary, $CH(P_i)$ must lie entirely on one side of the line common to $e_{Q'}$ (see Fig. 4). Similarly, $CH(P_2)$ must lie entirely on one side of the line common to $e_{Q''}$.

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

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

When $e_{Q'}$ or $e_{Q''}$ is not on the boundary of the convex hull of its unfolding, then naively 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 $(P^*,F^*)$ and attaching it to $(P_1,F_1)$ so that in the folded state $e_{Q'}$ becomes coincident to an edge on the boundary of the combined unfolding’s convex hull. We call $(P^*,F^*)$ a bridge and say that $e_{Q'}$ has been bridged to the boundary of the convex hull. As we will show in Sec. 4.

**LEMMA 2.** Given an unfolding $(P,F)$ and an edge $e_{Q'}$, it is always possible to bridge $e_{Q'}$ to the boundary of the convex hull. The bridge can be constructed in $O(M^2 + (N^2 + NM) \log(N + M))$ 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 $e_{Q'} \in (P_1,F_1)$ and $e_{Q''} \in (P_2,F_2)$, 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 $e_{Q'}$ and $e_{Q''}$ can always be bridged to the boundaries of the convex hulls of $P_1$ and $P_2$, respectively. Let $e_{Q'}^{new}$ be the bridge edge on the boundary of the convex hull that coincides with $e_{Q'}$ in the folded state, and similarly with $e_{Q''}^{new}$. Applying Lemma 1 to the modified $(P_1,F_1)$ and $(P_2,F_2)$ using $e_{Q'}^{new}$ and $e_{Q''}^{new}$ as the edges to join yields an unfolding $(P_3,F_3)$ that satisfies the conditions of Problem 2.

---

**Fig. 4 Two convex polygons placed next to each other are guaranteed not to intersect**
vex hull of the modified unfolding collapses onto $e_P$ such that in the folded state, an edge on the boundary of the convex hull is an edge on the boundary of $P$. 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
\[ O(M_i^2 + M_k^2 + (N_i^2 + N_1 M_1) \log(N_1 + M_1) + (N_k^2 + N_2 M_2) \log(N_2 + M_2)) \]

4 Constructing the Bridge

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

1. $e^f$ is on the boundary of $P$
2. $e^f$ 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 $e^f$.

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 $p = p_1 p_2 \ldots p_n$ in the order they appear in the chain.

4.1 Case 1: Edges on the Boundary. The procedure when $e^f$ is an edge on the boundary of $P$ is based on the following lemma.

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

Proof. Because $P$ is a polygon, its convex hull $CH(P)$ is also a polygon. The vertices of $CH(P)$ are a subset of the vertices of $P$. If the vertices on the boundary $V(P) = \{v_1^f, v_2^f, \ldots, v_n^f\}$ are numbered in clockwise direction, $CH(P)$ can be represented as an increasing sequence $(i_1, i_2, \ldots, i_n)$ such that $(v_{i_j}, v_{i_{j+1}}, v_{i_{j+2}})$ are the $n$ vertices of $CH(P)$ in clockwise order. If $e^f = (v_{i_j}, v_{i_{j+1}})$, let $P_{i_k}^CH$ be a polygon bounded by the path $p_k = v_{i_j} v_{i_{j+1}} \ldots v_{i_k} v_{i_j}$ where $i_k \leq j < i_{k+1}$. Since $e^f$ is an edge of $P$, it is a boundary edge of $P_{i_k}^CH$. The convex hull edge $(v_{i_{j+1}} v_{i_{j+2}})$ is also on the boundary of $P_{i_k}^CH$. Finally, $P_{i_k}^CH$ 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 $e^f$ to $CH(P)$ that does not intersect with $P$ or $CH(P)$ except at the terminal vertices must exist inside $P_{i_k}^CH$.

In order to bridge $e^f$ 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 nonintersecting folds with fold angles alternating between $\pi$ and $-\pi$) so that the edge at the end of the path (the convex hull edge) collapses exactly onto $e^f$ in the folded state. This is always possible by virtue of the following lemma.

Lemma 4. Given a starting edge $e^f$ and any simple path $p = p_1 p_2 \ldots p_n$ such that the first vertex $p_1$ is the midpoint of $e^f$, 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 $e^f$.

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 $e^f$ 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 Algorithm 1 and is illustrated in Fig. 5.

Algorithm 1: CREATEPLEATS($e^f$, $p$)

Data: $e^f = (v_1^f, v_2^f)$ = starting edge $p = p_1 p_2 \ldots p_n$ = path to follow

Result: unfolding ($P^{1P}, P^{2P}$) = pleats satisfying Lemma 4

// Beginning with $e^f$
1 $v_{i+1,1} \leftarrow v_{i+1}^f, v_{i+2} \leftarrow \bar{v}_i^f$
2 // Compute fold line locations
3 for $i = 1, \ldots, n - 1$
4 $\ell_i^f \leftarrow$ perpendicular bisector of segment $p_i p_{i+1}$
5 $v_{i+1,1} \leftarrow$ reflection of $v_i$ over $\ell_i^f$
6 $v_{i+1,2} \leftarrow$ reflection of $v_{i+2}$ over $\ell_i^f$
7 $e_i^f \leftarrow \left( \frac{1}{2} (v_{i+1} + v_{i+1,1}) \right), \left( \frac{1}{2} (v_{i+2} + v_{i+1,2}) \right)$
8 end
9 Remove intersections;
10 $P^{1P} \leftarrow$ polygon bounded by the path
11 $v_{1,1} v_{1,2} \ldots v_{n-1,1} \{e_i^f, (e_i^f, -\pi)\}$
12 $P^{2P} \leftarrow \left( (e_i^f, -\pi) \right)$

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 $\|e^f\|$. 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 the 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 quadrilaterals 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.

**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 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 Algorithm 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 \( n_e + n_i - 1 \) 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 \( P^e \). Naively checking every pair of edges on the boundary of \( P^e \) 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 (Algorithm 2), illustrated in Fig. 6.

**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. \( p \) ← path from \( e^p \) to CH\((P)\) (Lemma 3);
2. \((P^p, F^p) \leftarrow \text{CREATEPLEATS}(e^p)\);
3. \( P_{new} \leftarrow P \cup P^p; F_{new} \leftarrow F \cup F^p \cup \{(e^p, \pi)\};\)
4. Remove intersections;
5. \( e^p_{new} \leftarrow e_0 \), created during bridge construction in line 2;

4.1.1 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 \( P^e_{CH} \) satisfies this criterion.

The straight skeleton of \( P^e_{CH} \) is the union of trajectories of vertices of \( P^e_{CH} \) when the boundary of \( P^e_{CH} \) is shrunk in such a way that all edges retain their orientation and move toward the interior of \( P^e_{CH} \) at the same speed. The straight skeleton was introduced in Ref. [23] as an alternative to the medial axis, which can contain parabolic arcs when \( P^e_{CH} \) is nonconvex. As its name suggests, the straight skeleton contains only line segments. When \( p_a \) is the boundary of an \( n_e \)-gon, the straight skeleton is a tree with \( (n_e - 2) \) vertices and \( 2(n_e - 3) \) edges that partitions the polygon into \( n_e \) regions, each of which contains one arc of \( p_a \) [23]. It can be computed straightforwardly by simulating the polygon shrinking process in \( O(n_e \log(n_e)) \) 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 \( p = p_1p_2…p_{n_e} \) be this path, with the first vertex \( p_1 \) located at the midpoint of \( e^p \), intermediate vertices \( p_2, …, p_{n_e - 1} \) at vertices in the straight skeleton, and the final vertex \( p_{n_e} \) on the convex hull edge.

4.1.2 Line 2–3: Overlay Pleats. Overlay \( p \) with accordion-style pleats using Algorithm 1. For the last pleat, rather than following the procedure in lines 3–5 of Algorithm 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 Algorithm 1, \( e_{n_e} \) is then assigned a fold angle of \( \pi \), 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

![Fig. 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 \( P^e_{CH} \) (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.](Image)
prespecified. Since, however, \( p_{n-1} \) 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 \( e^p \).

4.1.3 Line 4: Remove Intersections With the Input Unfolding. Although the path \( p \) lies entirely in the free space, the pleats following \( p \) may 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.

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).

**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.  

**Algorithm 3: BRIDGEFROMFOLDLINE((P,F),e^p)\)**

<table>
<thead>
<tr>
<th>Data: ((P,F)) = input unfolding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: ((P_{\text{new}}, F_{\text{new}})) = unfolding containing ((P,F))</td>
</tr>
<tr>
<td>(e^p_{\text{new}}) = edge on (\text{CH}(P_{\text{new}})) that folds onto (e^p)</td>
</tr>
<tr>
<td>1 (e_b) = a boundary edge on (P);</td>
</tr>
<tr>
<td>2 ({(P_1, F_1), e^p_{\text{CH}}} \leftarrow \text{BRIDGEFROMBOUNDARY}((P,F), e_b));</td>
</tr>
<tr>
<td>3 ({(P_2, F_2), e^p_{\text{CH}}} \leftarrow \text{REFLECTFACES}((P_1, F_1), e^p_{\text{CH}}));</td>
</tr>
<tr>
<td>4 ({(P_3, F_3), e^p_{\text{new}}} \leftarrow \text{BRIDGEFROMBOUNDARY}((P_2, F_2), e^p_{\text{ref}}));</td>
</tr>
</tbody>
</table>

This case requires construction of the edge-adjacency graph (Fig. 10) of the unfolding \((P,F)\). This is a graph \((\mathcal{V}', \mathcal{E}')\) with vertices in \(\mathcal{V}'\) located at the midpoints of the edges in \(P\) and \(F\), and edges \(\mathcal{E}' = \{(v'_i, v'_j) \mid e_i, e_j \text{ lie on the boundary of the same face in } (P,F)\}\). The edge-adjacency graph has the following properties:
4.2.1 Line 1: Choose A Boundary Edge. Before constructing the bridge, it is necessary to choose where to attach it to the unfolding. Since \( e_o \) is not on the boundary of \( P \), it is impossible to connect the bridge at \( e_o \) without causing self-intersection of the final unfolding. Any boundary edge \( e_b \) can be used here. For our implementation, we choose \( e_b \) 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 \( e_o \) to \( e_b \) in the edge-adjacency graph \( G' \) and (2) the path from \( e_b \) to the boundary of the convex hull in the straight skeleton produced when calling Algorithm 2. Note that to calculate this cost, we generate the straight skeleton of every region of free space \( P^{(2)} \) in the unfolding's convex hull. There are \( O(N) \) such regions.

4.2.2 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 \( e_o \) and \( e_b \) cannot simply be reflected over \( e_b \) to make \( e_b \) 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 Algorithm 2. Let \( e_{CH} \) be the resulting edge on the convex hull boundary.

4.2.3 Line 3: Reflect Faces Between \( e_o \) and \( e_{CH} \). In order to construct a \( e_{ref} \) on the boundary of the unfolding, reflect all faces between \( e_o \) and \( e_{CH} \) over \( e_{CH} \). These faces can be found by consulting the edge-adjacency graph \( G' \). A path in \( G' \) from the vertex corresponding to \( e_o \) to the vertex corresponding to \( e_{CH} \) yields a set of faces that connect \( e_o \) and \( e_{CH} \). Since \( e_{CH} \) is on the convex hull boundary, the reflected faces will not intersect with the rest of the unfolding.

4.2.4 Line 4: Bridge \( e_{ref} \) to the Convex Hull Boundary. The result of line 3 is that \( e_{ref} \) 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 \( e_{ref} \). Thus, any material attached to \( e_{ref} \) is as if it were added at \( e_{ref} \).

Lemma 8. 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 Algorithm 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 \( O(M^2 + (N^2 + NM) \log(N + M)) \) 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 \( O(N^{7/11} + c) \) time. The closest boundary edge to \( e_o \) 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 Algorithm 2, which by Lemmas 7 and 8 produces an unfolding with \( O(N) \) vertices and \( O(N + M) \) edges in \( O(N^2 + NM) \log(N + M) \) time. Since the path from \( e_o \) 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 \( O(N^2 + (N^2 + NM) \log(N + M)) = O(N^2 + NM) \log(N + M) \), according to Lemma 8.

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

4.3 Summary. We have now provided enough detail to support Lemma 2:

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

Proof. Sections 4.1 and 4.2 detail how to construct a bridge when \( e_o \) is a boundary edge or \( e_o \) is a fold line. The edge to join \( e_o \) must fall into one of these cases. It is therefore always possible to bridge \( e_o \) to the convex hull.

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

5 Experimental Results

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 Algorithm 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)-(11(d) demonstrate Algorithm 3, when an edge to join is on the interior of the unfolding. The square pyramid in the unfolded state, the face lies flat against the surface of the square pyramid so that one side is doubly covered. Similarly, the insect connector 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 Algorithm 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 Algorithm 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 Algorithm 3 even if Algorithm 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 Sec. 1, depending on the application, certain folds should not be cut; however, a 3D 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.

Fig. 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.
7 Conclusion

This paper addresses automatic composition of unfoldings of 3D surfaces. We show that given the unfoldings of two 3D 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 Grant Nos. 1240383 and 1138967. C.S. was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

References