Origamizer: A Practical Algorithm for Folding Any Polyhedron

Erik D. Demaine[*1] **and Tomohiro Tachi**[†2]

1 **MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA**
edemaine@mit.edu

2 **Department of General Systems Studies, The University of Tokyo, Japan**
tachi@idea.c.u-tokyo.ac.jp

Abstract

It was established at SoCG'99 that every polyhedral complex can be folded from a sufficiently large square of paper, but the known algorithms are extremely impractical, wasting most of the material and making folds through many layers of paper. At a deeper level, these foldings get the topology wrong, introducing many gaps (boundaries) in the surface, which results in flimsy foldings in practice. We develop a new algorithm designed specifically for the practical folding of real paper into complicated polyhedral models. We prove that the algorithm correctly folds any oriented polyhedral manifold, plus an arbitrarily small amount of additional structure on one side of the surface (so for closed manifolds, inside the model). This algorithm is the first to attain the *watertight* property: for a specified cutting of the manifold into a topological disk with boundary, the folding maps the boundary of the paper to within ε of the specified boundary of the surface (in Fréchet distance). Our foldings also have the geometric feature that every convex face is folded seamlessly, i.e., as one unfolded convex polygon of the piece of paper. This work provides the theoretical underpinnings for Origamizer, freely available software written by the second author, which has enabled practical folding of many complex polyhedral models such as the Stanford bunny.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases origami, folding, polyhedra, Voronoi diagram, computational geometry

Digital Object Identifier 10.4230/LIPIcs.SoCG.2017.34

1 Introduction

The ultimate challenge in computational origami design is to devise an algorithm that tells you the best way to fold anything you want. Several results tackle this problem for various notions of "best" and "anything". We highlight two key such results, from SoCG'96 and SoCG'99 respectively. The tree method [6, 7, 4] finds an efficient folding of a given square of paper into a shape with an orthogonal projection equal to a scaled copy of a given metric tree. We use the term "efficient" because the method works well in practice, being the foundation for most modern origami design, but exact optimization of the scale factor (the usual measure of efficiency) is a difficult computational problem, recently shown NP-hard [3], but one that can be handled reasonably well by heuristics. The strip method [1] finds a folding of a given piece of paper into a scaled copy of any desired polyhedral complex (any

[*] Supported in part by NSF ODISSEI grant EFRI-1240383 and NSF Expedition grant CCF-1138967.
[†] Supported in part by JST PRESTO program and JSPS KAKENHI 16H06106.

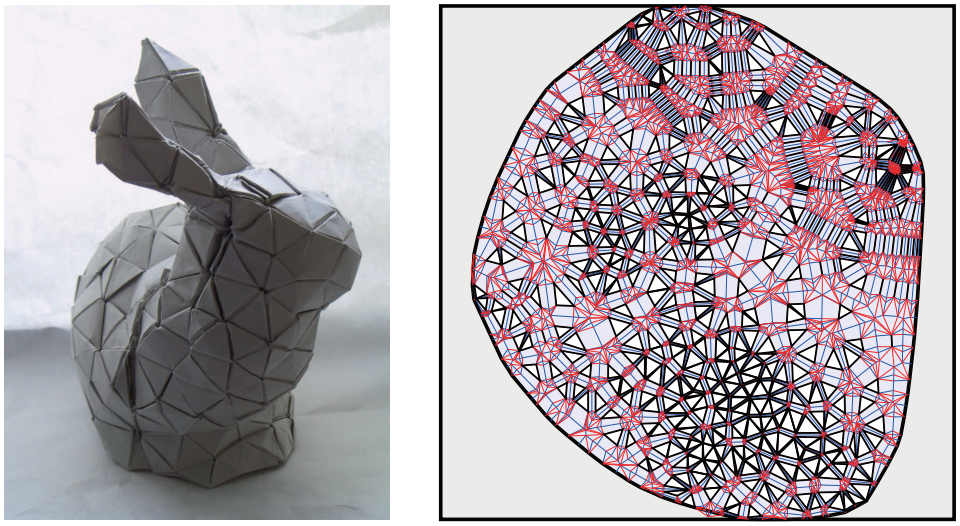


Figure 1 Origamizer software [8, 9] applied to 374-triangle Stanford bunny: real-world folding (left) and computed crease pattern (right).

connected union of polygons in 3D), which is a much broader notion of "anything". But it inherently provides no way to optimize the scale factor, forcing extreme inefficiency in paper usage (unless the piece of paper is a long narrow strip).

Our goal is to achieve the best of both these worlds. On the one hand, we want to fold arbitrary polyhedral complexes. On the other hand, we want to be able to optimize the scale factor to find efficient and ideally practical foldings. The vision is that the origami designer of the future uses 3D modeling software to design their target figure, gives it to an algorithm, and out comes a practical origami design.

We are not far from this vision today. *Origamizer* [8, 9] is freely available computer software implementing a relatively new heuristic for this problem. It achieves surprisingly practical foldings for complicated polyhedral models. For example, Figure 1 shows the result for the classic Stanford bunny, coarsened to 374 triangles. In this design, 22.3% of the paper area makes up the actual surface of the target shape – about a 2:1 scale factor in each dimension – which matches the material usage ratio in most practical origami design. The crease pattern is unlike most representational origami designs today, likely incapable of being designed by hand, yet it is also quite reasonable to fold in practice.[1]

The catch is that the heuristic implemented by the Origamizer software sometimes fails: sometimes it cannot even find a feasible solution, which makes it impossible to optimize even locally. In this paper, we develop an *Origamizer algorithm* that is guaranteed to find a feasible folding, for any orientable polyhedral manifold. In fact, we describe a large family of feasible foldings, with many free parameters, similar to the original Origamizer heuristic. In particular, Figure 1 falls within our family of foldings. The key new guarantee is that our family of foldings is nonempty, and that we can find such a folding (actually many) algorithmically. While we do not consider here how to optimize within the family of foldings (this is likely a computationally difficult problem, like the tree method), the starting points found by our algorithm open the door to a wealth of optimization heuristics. The result

[1] To see an accelerated video of the 10-hour folding process, visit `http://www.youtube.com/watch?v=GAnW-KU2yn4`

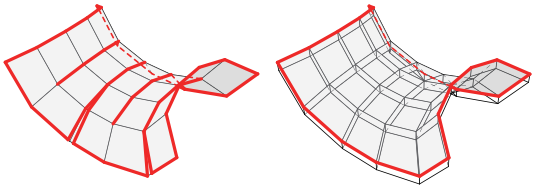


Figure 2 Folding a surface with gaps like the strip method (left) versus a watertight folding with some extra tiny facets like our method (right). The paper boundary is drawn thick.

should be at least as efficient as the existing Origamizer software, because the family of foldings is broader, but now it is also guaranteed never to fail. We plan to implement this provably correct algorithm in a future version of the Origamizer software.

Our Origamizer algorithm proves the existence of a new type of folding, called *watertight*, for any specification of where the boundary of the paper should go. That is, suppose we are told how to cut the given oriented polyhedral manifold into a topological disk with boundary. (If the manifold is itself a disk, no cutting is necessary.) Informally, a watertight folding has no holes, gaps, or slits internal to this boundary – only paper. Formally, the boundary of the

piece of paper (which can be any convex polygon) maps to within Fréchet distance ε of the boundary of the polyhedral surface, for a specified $\varepsilon > 0$. Thus the rest of the polyhedral surface must be covered entirely by the interior of the paper. By contrast, this property is violated violently in the strip method [1], which places the boundary of the paper on every face. Indeed, the lack of the watertight property seems a natural formalization of how the strip method felt like "cheating". Figure 2 shows a comparison.

Because the paper is homeomorphic to a disk, disk surfaces are the best we could hope to make watertight. Although we believe the watertight property is an essential feature of what makes Origamizer's foldings practical, it has one theoretical downside: it cannot fold exactly a desired polyhedral complex. By watertightness, a negative curvature vertex must be covered by an interior point of the piece of paper, which has a disk neighborhood. In other words, an entire neighborhood of a point of the piece of paper must fold to an entire neighborhood of a vertex of negative curvature. But such a folding is impossible by the Gauss-Bonnet Theorem.

Therefore Origamizer aims to fold a slight variation of the polyhedral complex, which adds small additional features at the vertices and along the edges. These features all have Hausdorff distance at most ε from the polyhedral complex, for any specified $\varepsilon > 0$. Furthermore, for orientable polyhedral surfaces, the features can all be placed on one side. In the case of an orientable closed polyhedron, like the Stanford bunny, we can place all of these features on the inside, so that they become invisible to the spectator. Such hidden features are standard in origami, and we believe they are well worth it to achieve the watertight property.

2 Problem Statement and Overview

The *Origamizer problem* is to find a convex polygon of paper P and a "watertight", "seamless", "ε-extra folding" of P into a given "polyhedral manifold" Q.[2] We need to define the four notions in quotes.

[2] Any convex polygon of paper can be folded into any other convex polygon after suitable scaling [1], so we can view the convex shape of the piece of paper as a free choice by the algorithm.

A *polyhedral manifold* Q is an embedded polyhedral manifold with strictly convex facets homeomorphic to a disk. Such Q could come from any polyhedral complex using standard techniques: for nonorientable or nonmanifold complexes, doubling every face to make them orientable manifolds; for orientable manifolds not homeomorphic to a disk, cutting each handle; and for nonconvex facets, subdividing into convex pieces. The polyhedral manifold Q has two specified sides, the *clean side* and the *tuck side*. For defining clockwise/counterclockwise orientations, we view the tuck side as the *top* side of the manifold. We require that the manifold does not touch itself, at least on the tuck side, other than the two boundary edges that come from each cut edge.

For any $\varepsilon > 0$, an *ε-extra folding* of a polygon of paper, P, into a manifold with specified boundary, Q, is a folded state of P (as defined, e.g., in [4, chapter 11]) whose image includes all of Q and otherwise is within ε of Q on the tuck side of Q. More precisely, we construct a *tuck-side ε offset* of Q, by unioning the portion of a radius-ε ball, centered at every point of Q, that lies on the tuck side of Q; when Q does not separate the ball into two portions (i.e., within ε of the original boundary of Q), we include the entire ball. Then the image of an ε-extra folding must lie entirely within the union of Q and its tuck-side ε offset. In particular, the Hausdorff distance between Q and the image of the folded state is at most ε.

Such a folding is *seamless* if the clean side of every facet of Q is covered by a single facet of the crease pattern of P, at the outermost layer of the folded state. Intuitively, this condition means that all visible creases and boundary edges of the piece of paper lie on edges and the tuck side of Q. In our foldings, we will satisfy the stronger property that each facet of Q is represented by a single uncreased face of paper, with no additional layers of paper even on the tuck side.

Such a folding is *watertight* if there is a closed curve C on P (the *effective boundary* of P) that folds to a 3D curve having Fréchet distance at most ε to the closed loop of boundary edges of Q, such that the folded image of the interior of C covers the interior of Q. In other words, the exterior of C is extraneous to the folding, and every point on the folded C is within distance ε of a corresponding point on the boundary of Q, where this correspondence proceeds monotonically around both curves (according to some parameterization). In our foldings, we will further guarantee that C is convex; indeed, C will be the boundary of the piece of paper P output by our algorithm.

Overview. Figure 3 gives a visual overview of the entire Origamizer algorithm. The algorithm first attaches ε-thin faces to the target polyhedral surface Q to form a target folded structure called a *waffle*, effectively splitting negative-curvature vertices into multiple positive-curvature *pockets* of the waffle [Section 3]. Next the algorithm locally "squashes" these pockets into the piece of paper, with angles large enough that they can be folded down to the desired angles [Section 4]. All that remains is to fold away the excess material between these squashed pockets, and thereby form the waffle. To guide this process, the algorithm first draws *streams* (smooth constant-width channels) that connect together corresponding edges and vertices of different squashed pockets [Section 5].

Ultimately, Origamizer uses a *Voronoi diagram* as the basis for its crease pattern: for any set of sites, we show how to fold an abstract waffle (not necessarily embedded in 3D) with exactly one pocket for every Voronoi cell [Section 7]. The challenge is to choose sites defining the Voronoi diagram so that the resulting abstract waffle can be folded into the desired waffle. The algorithm places one site at each squashed pocket, several sites along each stream, and additional sites to fill the rest of the paper [Section 6]. The resulting abstract waffle can be folded into the desired waffle by collapsing each stream's pockets to bring together the pockets at either end of the stream, and by collapsing all additional pockets [Section 7].

Given the page limit, we focus here on high-level sketches and figures of the required properties and algorithms. Refer to the full version of the paper [5] for the details and proofs.

3 Tuck Proxy and Waffles

Given the target polyhedral manifold Q, the first step of the algorithm computes a "tuck proxy" T, which is a special type of "polyhedral waffle". The *tuck proxy* T is a polyhedral complex that will contain our folding and which contains (and lies very close to) the target polyhedral surface Q. Roughly speaking, a *polyhedral waffle* consists of *floor* polygons, which together form a topological disk, and *wall* polygons – wall *quadrangles* glued along floor edges, and wall *triangles* glued at floor vertices. The *top* edges of the wall polygons, opposite their attachment to their floor, must form an edge-2-connected planar graph called the *waffle graph*; refer to Figure 4. The *waffle dual* is the "modified dual" of the waffle graph. Roughly speaking, the *modified dual* is the usual planar dual plus a boundary node and incident edge for each edge of the outside face. (Thus, each boundary node has exactly one incident edge.)

In the tuck proxy, the floor polygons must be exactly the facets of Q. Any single floor vertex, floor edge, or floor polygon, together with its incident wall polygons, must form a polyhedral manifold homeomorphic to a disk (called a *waffle pocket*) that is intrinsically convex, meaning that no point has more than 360° of material. Furthermore, every wall polygon must have height at most ε, so that the tuck proxy is within ε of Q. In addition, the algorithm outputs a parameter ε' with $0 < \varepsilon' < \varepsilon$ that lower bounds how far the tuck proxy extends beyond Q, which guarantees no global self intersection up to that distance.

Figure 5 illustrates this step of the algorithm. The main idea is to inset the edges of Q on the tuck side (like the beginning of the formation of the 3D straight skeleton) by an amount ε' small enough that no collision events occur. This insetting bisects all dihedral angles, so in particular, it divides every reflex dihedral angle into two convex dihedral angles. This consequence is the key to how we guarantee that the waffle pockets are intrinsically convex. We then connect these edge offsets around each vertex of Q, which is equivalent to drawing a connected graph on a small sphere around the vertex. We first connect these offsets by a cycle on the sphere. The pocket formed by the floor polygon, two offset edges, and one edge of the cycle has a perimeter of at most 360° because of the strict convexity of the incident polygons and strict convexity of the angle between the offset and the floor. Then, to make the remaining pocket intrinsically convex, we subdivide the cycle by overlaying a regular tetrahedron and triangulating (if necessary). The resulting faces have edge lengths of at most 109.5°, so perimeter at most $328.5° < 360°$.

4 Waffle Pocket Squashing

Given the tuck proxy T and parameter ε', the next step of the algorithm computes a *local squashing* of each waffle pocket of the tuck proxy. Roughly speaking, local squashing consists of adding material between polygons in the waffle pocket until they lie flat and nonoverlapping in the plane. More formally, a local squashing draws each floor and wall polygon of the waffle pocket in the plane subject to only increasing the bottom angles of wall polygons, preserving convexity of the wall polygons, preserving the connectivity between the polygons, leaving no angular gaps between polygons at floor vertices, keeping the squashed wall polygons within ε' of the floor vertex/polygon, and preserving the cyclic order of the wall polygons around the floor vertex/polygon. The top edges of the squashed wall polygons must form a convex polygon in the plane, called the *cell*, so that the *cell perpendiculars* (normal to each cell edge,

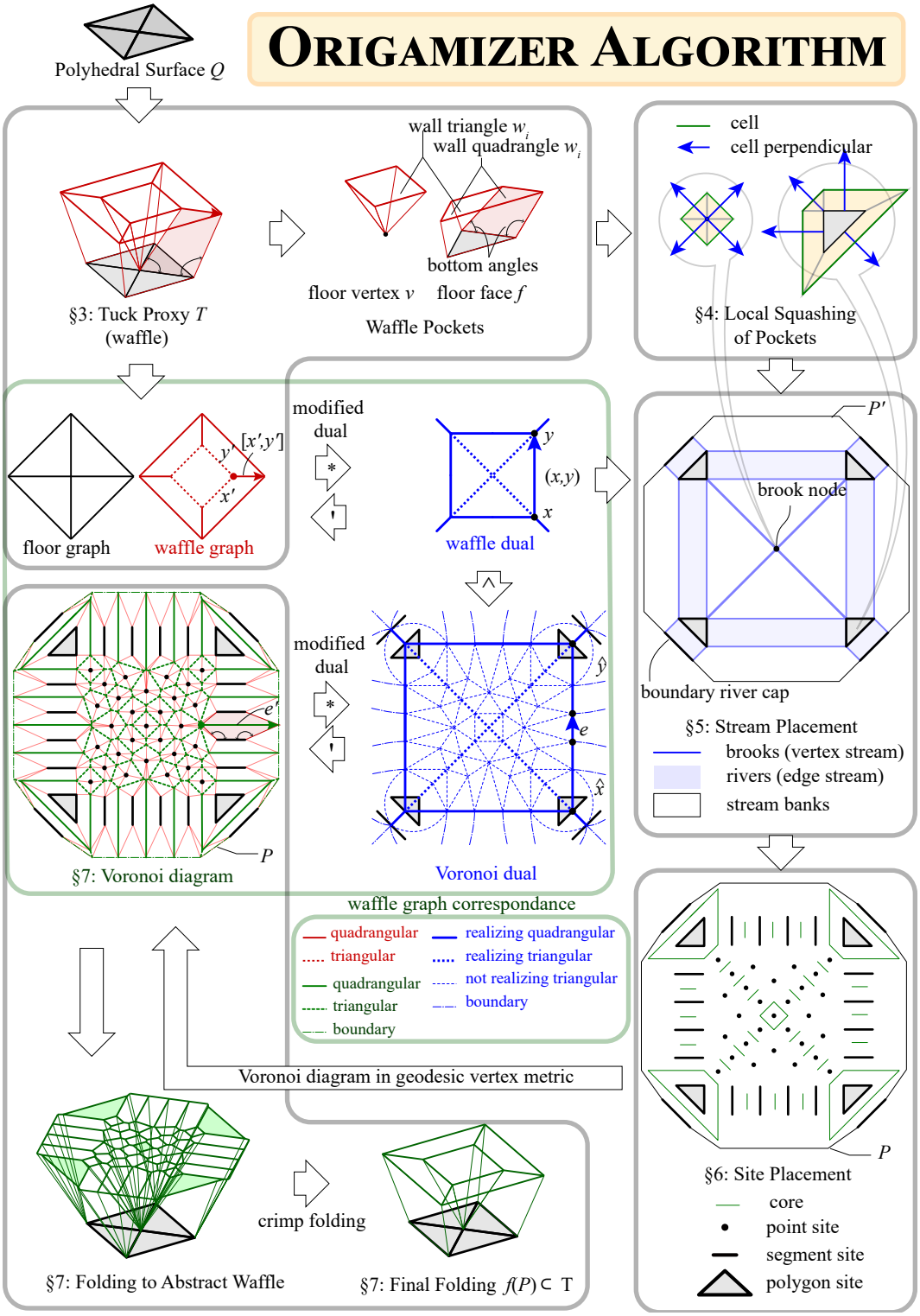


Figure 3 Visual overview of entire Origamizer algorithm, including intermediate data structures.

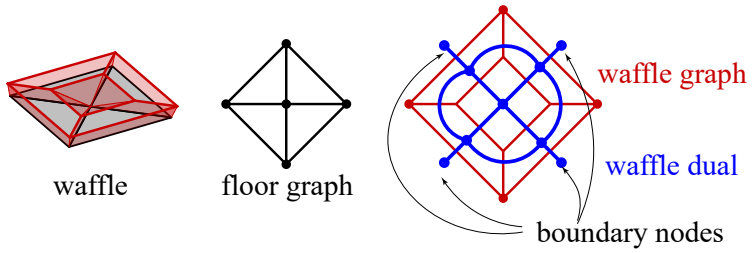


Figure 4 A waffle, the waffle graph, and the waffle dual. Wall faces are red; floor faces are grey.

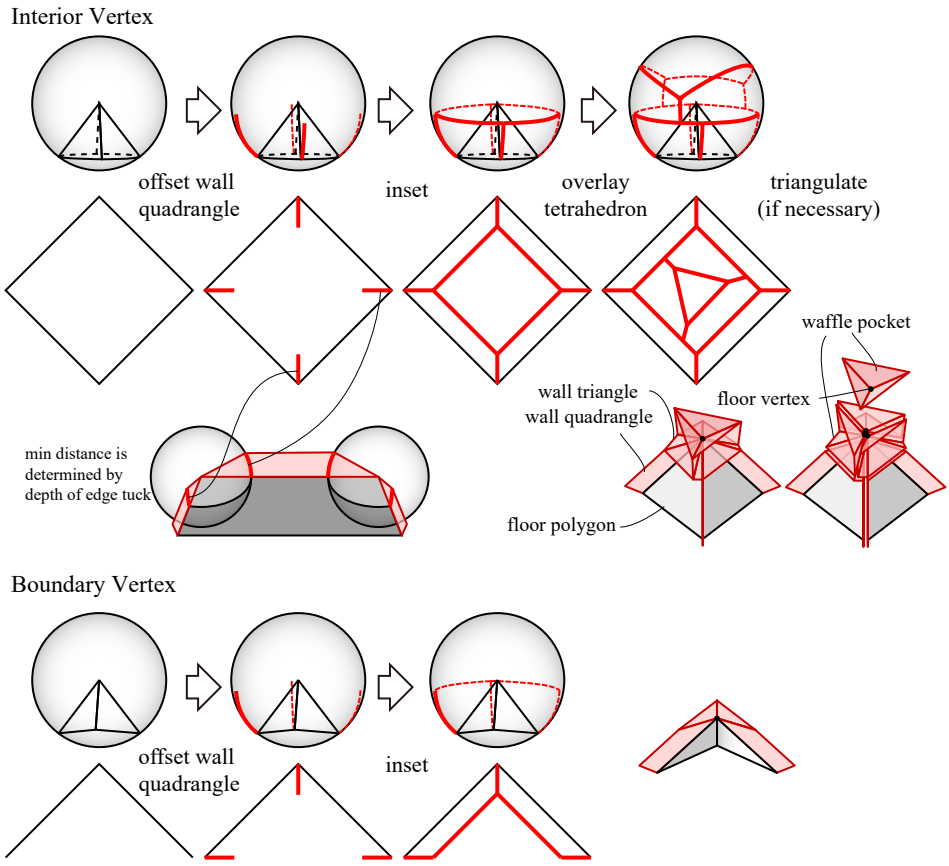


Figure 5 Construction of the tuck proxy for an interior vertex (top) and boundary vertex (bottom). The interior-vertex construction consists of three rows. Top: spherical view of behavior around a vertex; Middle: planar projection of behavior on the sphere; and Bottom: broader 3D view, around an edge and its two endpoints (left) and actual tuck proxy around the vertex (right).

and starting from the midpoint of the bottom of the corresponding squashed wall polygon) proceed counterclockwise around the floor vertex/polygon.

Figures 6 and 7 illustrate the two cases of this step of the algorithm, which get applied to all waffle pockets of the tuck proxy corresponding to floor vertices and floor polygons, respectively. The algorithm first vertex-unfolds [2] the wall polygons into the plane, leaving angular gaps evenly distributed between squashed wall polygons. We then place rays for each gap from the vertex, such that consecutive rays sandwich the squashed wall polygons and form an angle strictly less than 180°. (Specifically, each ray is the angular bisector between

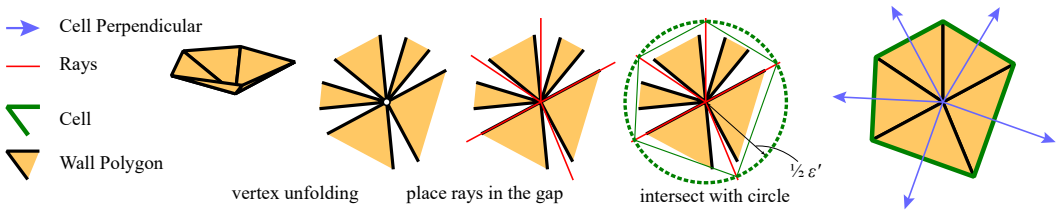


Figure 6 Local squashing algorithm for a waffle pocket corresponding to a floor vertex.

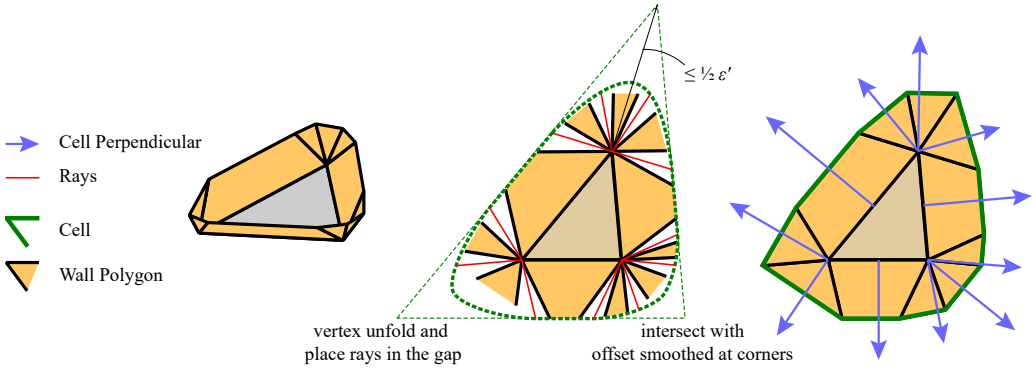


Figure 7 Local squashing algorithm for a waffle pocket corresponding a floor polygon.

the angular bisectors of two consecutive squashed wall polygons, possibly rounded to one of those wall polygon's boundaries.) We connect the intersection of the rays with a smooth convex curve (for the floor vertex case, a circle; and for the floor polygon case, the offset of the floor polygon smoothed at the corner with quadratic Bézier curve), placed close enough to and surrounding the floor vertex/polygon, to obtain a strictly convex cell. The resulting squashed wall polygons are bounded by the rays and the cell, and thus the bottom angles only increase. Because we use an offset curve to intersect the rays, wall quadrangles attached to a floor polygon squash into trapezoids.

5 Placing Streams

Given the tuck proxy T, parameter ε', and a local squashing of its waffle pockets, the next step of the algorithm computes a *stream placement*, which consists of a convex polygon P' and a mapping from various features of the waffle dual into geometric structures drawn on P'. Roughly speaking, each waffle dual node maps to either a point in P' (for a waffle pocket corresponding to a floor vertex) or an isometric embedding of a facet of Q in P' (for a waffle pocket corresponding to that floor polygon); and each waffle dual edge maps to a *stream* – a C^1 curve consisting of line segments and circular arcs, possibly *thickened* orthogonally. Specifically, if the waffle dual edge corresponds to a wall quadrangle, then the stream connects two equal-length edges of the placed floor polygons, and the stream is thickened by an amount equal to that common edge length, forming a *river* (as in [6, 7]). On the other hand, if the waffle dual edge corresponds to a wall triangle, then the stream connects two points, either placed floor vertices or vertices of placed floor polygons, and the stream has zero thickness, forming a *brook*.

This step of the algorithm also outputs a number $\delta > 0$ that is a lower bound on the "clearance" of the output structures, that is, the critical radius at which a disk Minkowski-

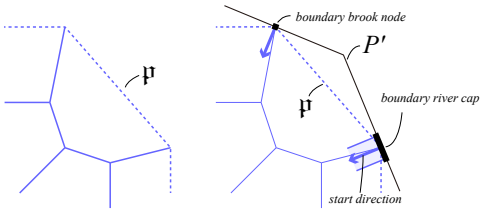


Figure 8 Left: Tutte embedding with outside face $\mathfrak{p}$. Right: Construction of start ray and convex polygon P'.

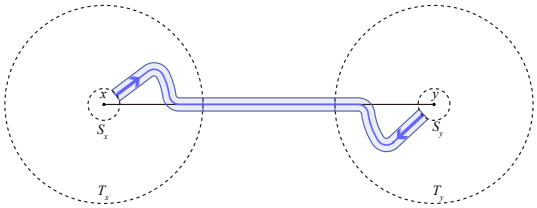


Figure 9 Connecting two embedded waffle dual nodes x and y with a river.

summed with the structures causes a collision event. This lower bound is important for bounding the number of creases in the ultimate Origamizer design.

This step of the algorithm consists of two major parts. In the first part, we embed the waffle dual in the plane using a Tutte embedding [10], and construct a convex polygon P' so that the boundary nodes lie on the boundary of P'; refer to Figure 8. During this construction, we guarantee that every boundary node attached to a river has enough clearance to be thickened parallel to its edge of P', to its desired width, without leaving that edge of P' and without intersecting other boundary nodes.

In the second part, we place floor vertices and polygons at the nodes of the embedded waffle dual graph, and connect them by rivers and brooks, respectively, along the edges of the embedded graph; see Figure 9. The main challenge is that the directions of the edges of the embedded waffle dual graph do not (in general) match the start directions of the streams defined by the cell perpendiculars of the local squashing. We construct a local structure around each embedded waffle dual node to adjust these directions without self-intersection. Specifically, this local structure consists of three nested disks, centered at each waffle dual node x and having radii $r_x < s_x < t_x$, each helping to adjust the directions of streams incident to x; refer to Figure 10. The radius-r_x disk R_x separates the streams from the floor vertex/polygon; the radius-s_x disk S_x bends the streams to meet the disk boundary orthogonally; and the radius-t_x disk T_x twists the streams to match the directions of the incident edges of the embedded waffle dual graph, while carefully avoiding collisions by having k tracks for a degree-k vertex x. In the last twisting step, we rotate the embedded floor polygon so that one edge normal does not require twisting, conceptually cut the disk there, and look at the $T_x - S_x$ annulus in polar view; then we make each connection from inside (p_i) to outside (q_i) in counterclockwise order, using the outermost unused track for counterclockwise (leftward) connections and the innermost unused track for clockwise (rightward) connections. To make the streams C^1 and obtain positive clearance δ, we *fillet* each corner of these streams, replacing each sharp corner by a circular arc of sufficient radius. To guarantee that these disks are local to their corresponding nodes, we scale up the Tutte embedding by a sufficient (but finite) factor.

6 Placing Sites

Given the output from the previous three steps (the tuck proxy T and parameter ε' from Section 3, a local squashing of waffle pockets from Section 4, and a stream placement from Section 5), the next step of the algorithm computes a *site placement*: the final piece of paper P, and a set of point, segment, and polygon *sites* on P. This site placement satisfies several properties which we state in terms of their generalized Voronoi diagram. The Voronoi diagram we use is in the *geodesic vertex metric*, which measures the geodesic (shortest-path)

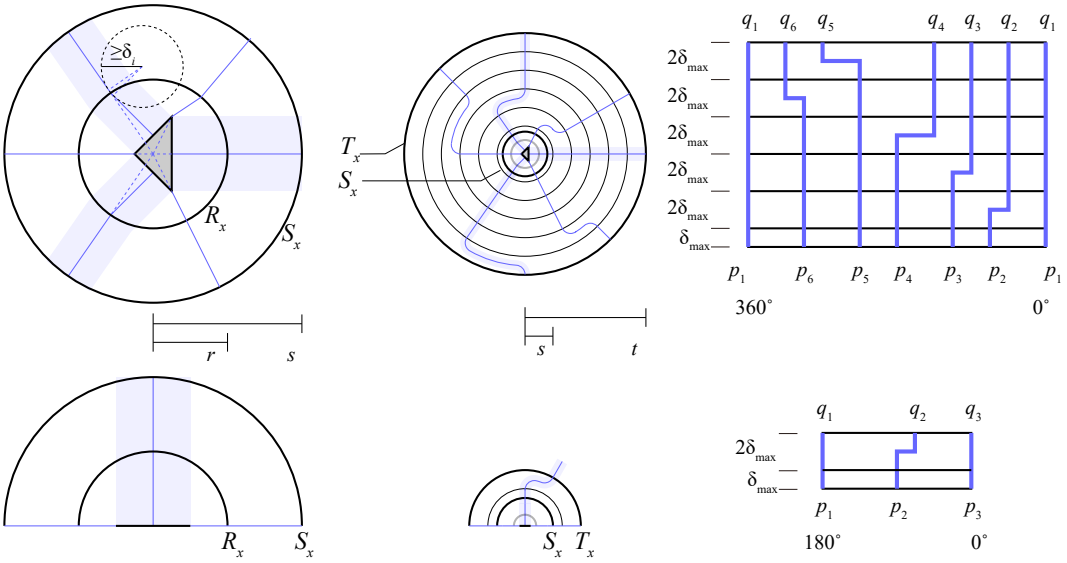


Figure 10 Attaching rivers to a facet of Q. Top: interior case. Bottom: boundary case. Left: filling disks R_x and S_x of radii $r_x < s_x$. Middle: filling the annulus $T_x \setminus S_x$ of outer radius $t_x > s_x$. Right: polar view.

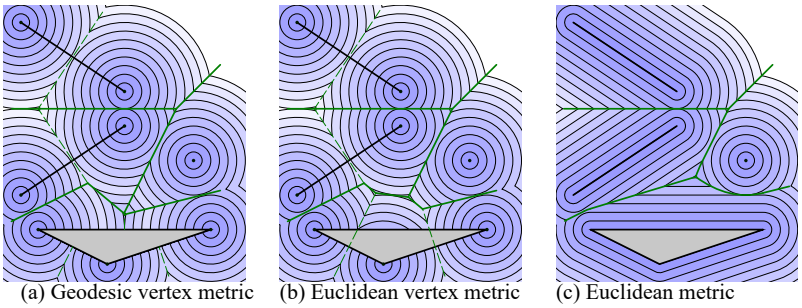
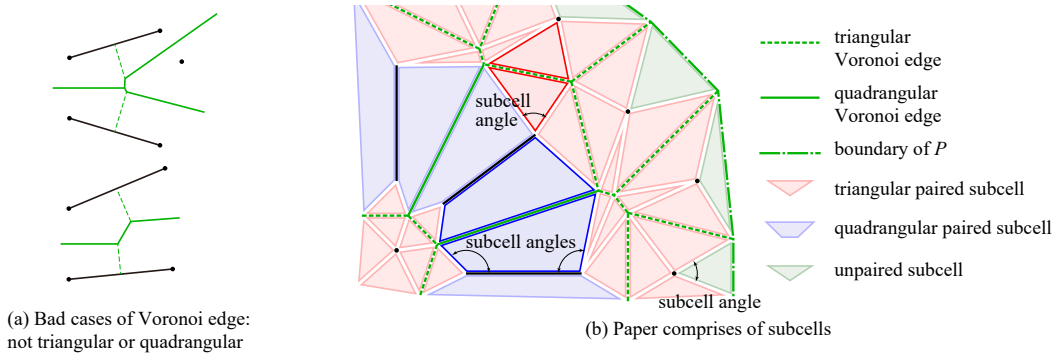


Figure 11 The geodesic vertex metric and the resulting Voronoi diagram of point, segment, and polygon sites, compared with the usual Euclidean metric (where we measure the minimum distance to any point of a site) and an intermediate "Euclidean vertex metric" (where we measure the minimum Euclidean distance to a vertex of a site).

distance between a point of the paper and the nearest vertex of a site, viewing all sites as planar obstacles that cannot be crossed (and thus must be routed around) by a shortest path. Refer to Figure 11.

The site placement requirements are the following:

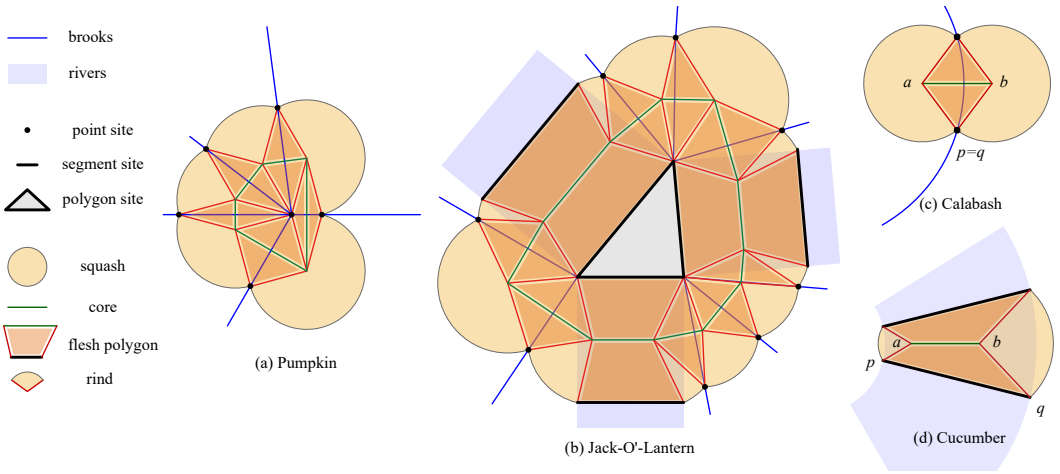
1. The Voronoi diagram can be contracted into the waffle graph of T, i.e., the modified dual of the Voronoi diagram is a supergraph of a subdivision of the waffle dual (Figure 3 middle). Thus each edge of the waffle dual is *realized* by a path of edges in the Voronoi dual, and the corresponding waffle graph edges *realize* the corresponding Voronoi edges.
2. Each Voronoi edge is a straight segment which mirror-reflects its defining site vertices (avoiding cases like Figure 12(a), which can generally happen when using the geodesic vertex metric). As shown in Figure 12(b), we obtain a mirror-reflected pair of triangles or quadrangles called *paired subcells*, where a paired subcell is quadrangular if and only if the Voronoi edge realizes a wall quadrangle.



(a) Bad cases of Voronoi edge: not triangular or quadrangular

(b) Paper comprises of subcells

Figure 12 Triangular and quadrangular edges, subcells, and subcell angles.



(a) Pumpkin

(b) Jack-O'-Lantern

(c) Calabash

(d) Cucumber

Figure 13 Different types of squashes generated by the site placement algorithm: pumpkins around brook nodes, jack-o'-lanterns around placed facets, calabashes along brooks, and cucumbers along rivers.

3. Each Voronoi edge realizing an edge of the waffle graph must have subcell angles (formed with the site) that are at least the bottom angles of the corresponding wall polygons.
4. Every point of the paper P is within ε' (Euclidean) distance of a site vertex.

To guarantee these properties of the Voronoi diagram of the placed sites, we also place a collection of (open set) planar regions called *squashes*, of four different types (see Figure 13): *pumpkins* at brook nodes (points connecting multiple brooks), *jack-o'-lanterns* around placed floor polygons, *calabashes* along brooks, and *cucumbers* along rivers. The sites defining a squash lie on the boundary of the squash. Within each squash, there is a *core*, which is part of the Voronoi diagram of the generating sites, and *flesh polygons*, which are mirror-reflected triangles and quadrangles between sites and the core. We design so that the flesh polygons have angles (against the sites) that are at least the bottom angles of corresponding wall polygons.

A key property is that each squash is the continuous union of geodesic (open) disks centered at points on the core and passing though the nearest vertices of the generating sites. This property ensures that, if there are no other sites in the squash, then the core is guaranteed to be part of the global Voronoi diagram, and thus the flesh polygons will be contained in paired subcells. Thus the algorithm places sites and squashes tightly enough

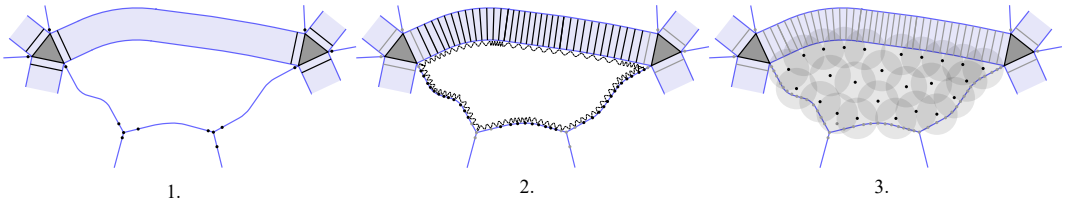


Figure 14 Site placement algorithm overview. 1. Adding sites at brook nodes and placed floor faces. 2. Adding point sites along brooks and segment sites along rivers. 3. Filling stream banks with point sites.

(so any point of P has distance at most ε' to a site) while using the union of squashes as a protective region in which we forbid placing any new sites. Specifically, the site-placement algorithm consists of following three steps; refer to Figure 14.

1. **Node sites:** Add a pumpkin at each brook node, and a jack-o'-lantern at each placed floor polygon. The core of each pumpkin and jack-o'-lantern is exactly the cell of the local squashing of the corresponding waffle pocket.
2. **Stream sites:** Place a sequence of point sites along each brook, and calabashes between consecutive pairs of placed point sites, such that the angle of each flesh polygon equals the bottom angle of the wall triangle of T corresponding to the brook. Place a sequence of segment sites along each river, and cucumbers between consecutive pairs of placed segment sites, such that the angles of each flesh polygon equal the bottom angles of the wall quadrangle of T corresponding to the river. Because streams are (thickened) line segments and circular arcs, we can design calabashes and cucumbers to have mirror symmetry between consecutive sites along each stream. If the gap between two consecutive sites is too big, other streams (and thus stream sites) might intersect the squash. In this case, we subdivide by bisecting the gap, adding an additional site along the stream, which converges to squashes intersecting only the streams they belong to (by the smoothness and positive clearance δ of streams obtained in Section 5). Also subdivide sufficiently so that each point in the squash has distance at most $\frac{1}{2}\varepsilon'$ from the core.
3. **Bank sites:** Repeatedly add a point site wherever there is a point ε' away from the closest site. Here we use that squashes are contained in the Minkowski sum of each site with an ε'-radius disk, so that this process terminates without placing sites on squashes.

We claim that the resulting site placement gives a Voronoi diagram that can contract to the waffle graph of T. This claim follows because, for each stream following the modified dual of the waffle graph of T, there is a sequence of sites that are adjacent to each other through Voronoi edge containing the cores of calabashes or cucumbers, which are sandwiched by flesh triangles or quadrangles, respectively. In the Voronoi dual, such a sequence realizes an edge of waffle dual. (Refer to the waffle graph correspondence in the middle of Figure 3.)

The piece of paper P is defined to be the convex hull of the paired subcells (within P'), or equivalently, the convex hull of the sites and the Voronoi diagram (clipped to P'). Polygon P differs only slightly from P' from the previous step, possibly removing "nonsubcell" portions of P' near its vertices. There may still be regions of P that are not in any paired subcell, which we call *unpaired subcells*; see Figure 12(b). Unpaired subcells are all triangles, with two edges defined by legs of adjacent paired subcells and one edge defined by the boundary of P.

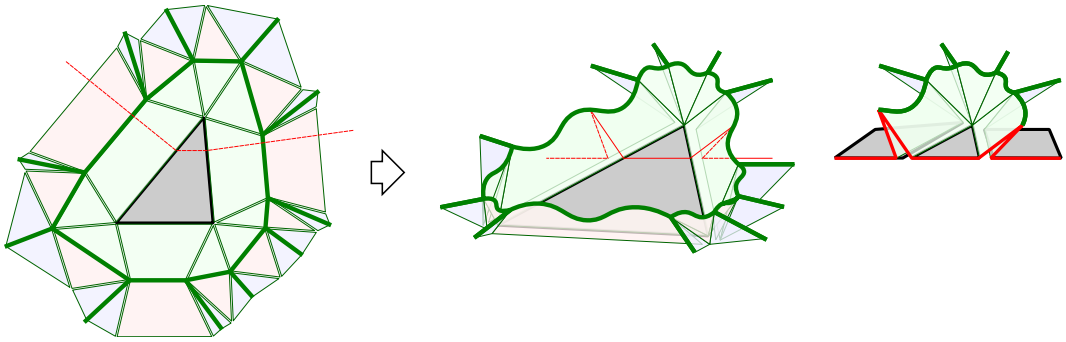


Figure 15 Folding each Voronoi cell (left) into a pocket of an abstract waffle (middle). Each wall is double covered by a pair of mirror-reflected paired subcells as shown in the cross section (right).

7 Voronoi Folding

Given the tuck proxy T and the site placement from Section 6, the final step of the algorithm computes a watertight seamless ε-extra folding of the piece of paper P into Q. In particular, the computed folding contains all facets of Q and lies on the tuck proxy T. We construct the folding by a sequence of *folding steps*, where each folding step treats the folded image resulting from previous steps as the "sheet of paper" to start from (never separating layers of paper that have been brought together by previous steps). In fact, each folding step produces an abstract metric polyhedral complex called an *abstract waffle*, which is topologically equivalent to a waffle and has an intrinsic metric for each floor and wall polygon. The last folding step's abstract waffle embeds directly on the tuck proxy, and thus we can realize the abstract structure isometrically in 3D. Validity of each folding step guarantees a consistent layer ordering in the final folded state (without paper crossing itself).

Initial folding step into abstract waffle: Define the graph G to consist of the following edges: the Voronoi edge of every paired subcell (within P) and the boundary edge of every unpaired subcell. The initial folding step folds P into an abstract waffle A whose waffle graph is G; refer to Figure 15. Specifically, for every Voronoi edge of G, the two paired subcells of P (which are reflections of each other) fold onto each other to doubly cover the corresponding wall polygon of A; and for every boundary edge of G, the unpaired subcell of P fold to singly cover the corresponding wall polygon of A. This folding gives a consistent metric to every wall polygon of the abstract waffle A. In particular, each cell of the Voronoi diagram (within P) becomes a pocket of A.

The floor polygons of A are the placed floor polygons in P. Because the Voronoi diagram realizes the waffle graph of the tuck proxy T, the floor polygons in A are connected together in the same way as the floor polygons in T. In other words, the floor of A is isometric to Q.

The remaining challenge is that A has excess wall polygons and also larger angles compared to T. The main idea of the following steps is to remove excess wall triangles that do not realize the waffle graph of T, reduce the bottom angles of walls realizing the waffle graph of T, and then glue isometric walls together to have a simplified topology. Each of the following folding steps involves folding one or two creases that radiate from a single floor vertex, so they can easily be viewed as foldings of the 1D waffle graph, where edge lengths represent bottom angles and each quadrangular edge is split into two subedges to represent its two bottom angles.

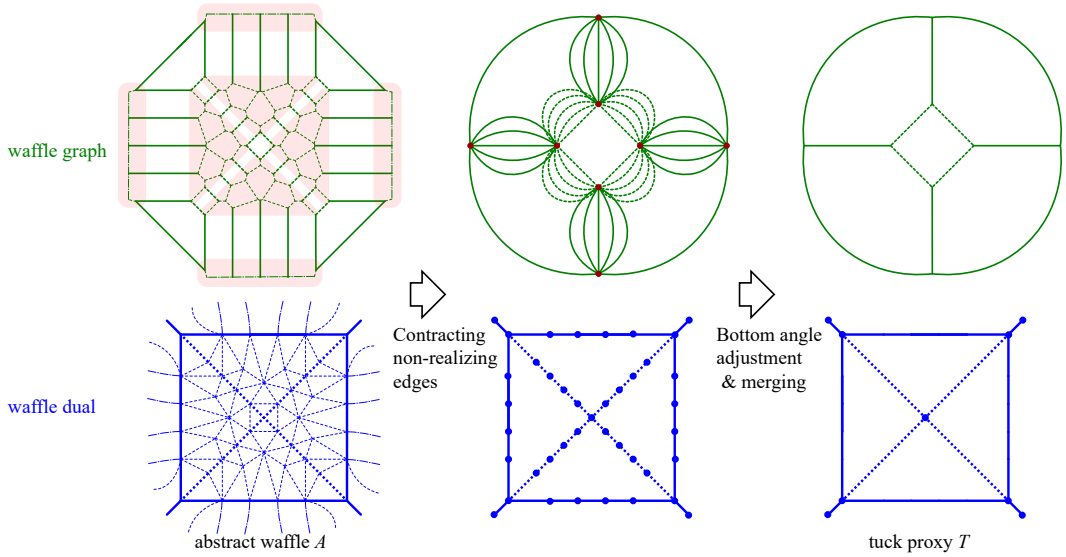


Figure 16 The waffle graph of the abstract waffle, reduced to the waffle graph of the tuck proxy.

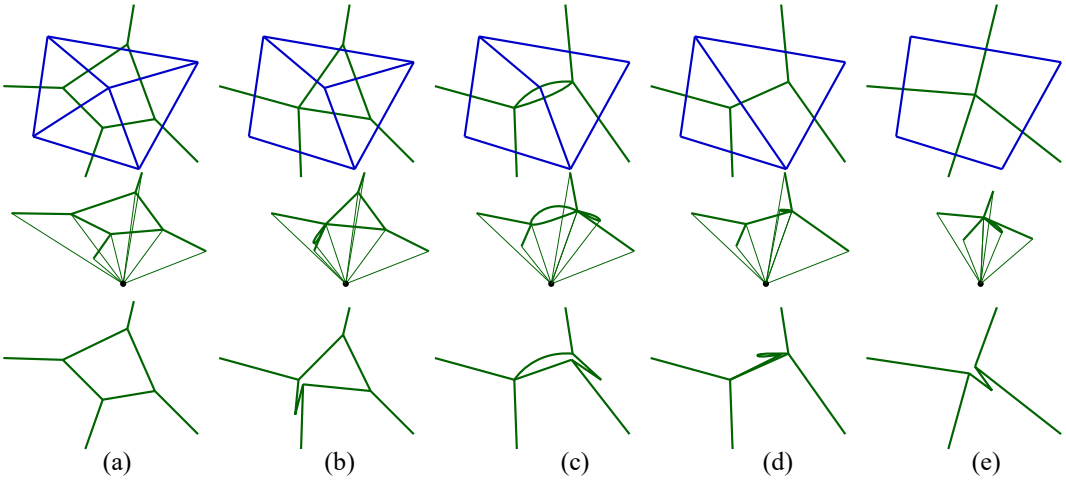


Figure 17 Contraction in Voronoi diagram / deletion in Voronoi dual to form the waffle graph / waffle dual of the tuck proxy.

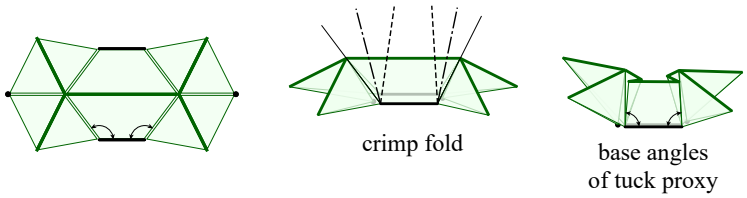


Figure 18 Angle adjustment to fit quadrangular wall of T.

Contracting and merging nonrealizing edges: First we contract each edge of the Voronoi diagram that does not realize an edge of the waffle graph of T (Figure 16 from left to middle). Each edge contraction can be achieved by folding the corresponding wall triangle in half (along an angular bisector) and gluing it against an adjacent wall polygon, possibly wrapping around the walls of the same pocket (Figure 17, (a) $\to$ (b) and (b) $\to$ (c)). If we ever have multiple edges connecting the same two nodes, we can unify the angles to the smallest angle by crimp folding the larger angles (Figure 17, (c) $\to$ (d)), making these multiple edges isometric to each other, and thus the wall polygons can be stacked on top of each other. This fused edge can be contracted again (Figure 17, (d) $\to$ (e)) to collapse a pocket. By applying these folding steps to all edges not realizing the waffle graph of T, we obtain a simplified waffle graph whose dual consists of just the paths of edges realizing each edge of the waffle dual of T.

Merging realizing edges: Each dual path corresponds to a sequence of wall polygons that represent the same edge in the waffle graph. To fuse them together, we crimp their bottom angles to match the bottom angles of the corresponding wall polygons of T; see Figure 18. Once they have the same base angles, we can stack the wall polygons on top of each other and regard them as a single wall polygon (Figure 16 from middle to right). Now each wall polygon can be isometrically embedded into the corresponding wall polygon of T, which gives the resulting folded state being a subset of T. Therefore, the final folded state f is the desired ε-extra folding of Q.

Remaining desired properties: The seamless property follows because the floor polygons are isometric to Q, and no other part of the paper folds strictly interior to any facet of Q. Watertightness follows by considering a point p moving along the boundary of P in counterclockwise order and a point q moving along the boundary of Q in counterclockwise order. The correspondence between p and q is given by (1) if p is on a boundary site, then $f(p) = q$; and (2) if p is between two consecutive boundary sites, then q stays at the corresponding vertex, which stays within distance ε from $f(p)$.

References

1. Erik D. Demaine, Martin L. Demaine, and Joseph S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Computational Geometry: Theory and Applications*, 16(1):3–21, 2000.
2. Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. Vertex-unfolding of simplicial manifolds. In *Discrete Geometry: In Honor of W. Kuperberg's 60th Birthday*, pages 215–228. Marcer Dekker Inc., 2003.
3. Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. In *Origami^5: Proceedings of the 5th International Conference on Origami in Science, Mathematics and Education*, pages 609–626. A K Peters, Singapore, July 2010.
4. Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.
5. Erik D. Demaine and Tomohiro Tachi. Origamizer: A practical algorithm for folding any polyhedron. Manuscript, 2017. URL: `http://erikdemaine.org/papers/Origamizer/`.
6. Robert J. Lang. A computational algorithm for origami design. In *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 98–105, Philadelphia, PA, May 1996.

7 Robert J. Lang and Erik D. Demaine. Facet ordering and crease assignment in uniaxial bases. In *Origami*^{4}*: Proceedings of the 4th International Conference on Origami in Science, Mathematics, and Education*, Pasadena, California, September 2006.

8 Tomohiro Tachi. Software: Origamizer, 2008. URL: `http://www.tsg.ne.jp/TT/software/`.

9 Tomohiro Tachi. Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):298–311, 2010. `doi:10.1109/TVCG.2009.67`.

10 W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–767, 1963. URL: `http://plms.oxfordjournals.org/cgi/pdf_extract/s3-13/1/743`, `doi:doi:10.1112/plms/s3-13.1.743`.