Planar Drawings of Origami Polyhedra

Erik D. Demaine   Martin L. Demaine
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
{eddemaine,mldemaine}@uwaterloo.ca

Abstract

We present a new infinite class of polyhedra based on a class of origami bases that we have developed. To understand these polyhedra and their underlying bases, we examine drawings of their graphs. We present an elegant linear-time algorithm to find a straight-line planar drawing. It displays a recursive structure in the polyhedra that may lead to interesting fractals. We introduce a "zoom" feature that allows one to interactively explore the details of the graph while displaying this recursive structure.

1 Introduction

Origami mathematics is the mathematical study of the properties of origami (paper folding). The area of origami mathematics is still in its infancy, having only been seriously studied for the past seventeen years [12]. In this paper, we study properties of origami through the use of graph drawings of origami polyhedra.

We hope to discover necessary and sufficient conditions for when a collection of folds can be used to produce a flatly folded sheet of paper. Using origami terms, we want to know when a crease pattern permits a flat origami. A characterization of this would be of interest to all origami mathematicians. Hull [11] has studied this question, but useful sufficient conditions are not yet known. Unfortunately, the problem is known to be $\mathcal{NP}$-hard [2]. However, this does not prevent a characterization of flat-foldable crease patterns, only the tractability of such a characterization.

One conjecture that we have arrived at from this work is that every flat-foldable crease pattern has a Hamiltonian cycle. This property would not be sufficient, since there are Hamiltonian crease patterns that satisfy other known necessary conditions and are not flat foldable. However, a similar condition that is $\mathcal{NP}$-hard to verify but easy to understand may fill the gap in understanding flat origami.

Origami *bases* are the starting point for origami models. Lang has algorithmically defined the crease patterns for the largest class of bases, called *uniaxial bases.* A uniaxial base has the properties that the faces are perpendicular to a plane P, the intersection of the base with P equals the projection to P of the rest of the base (which must be above

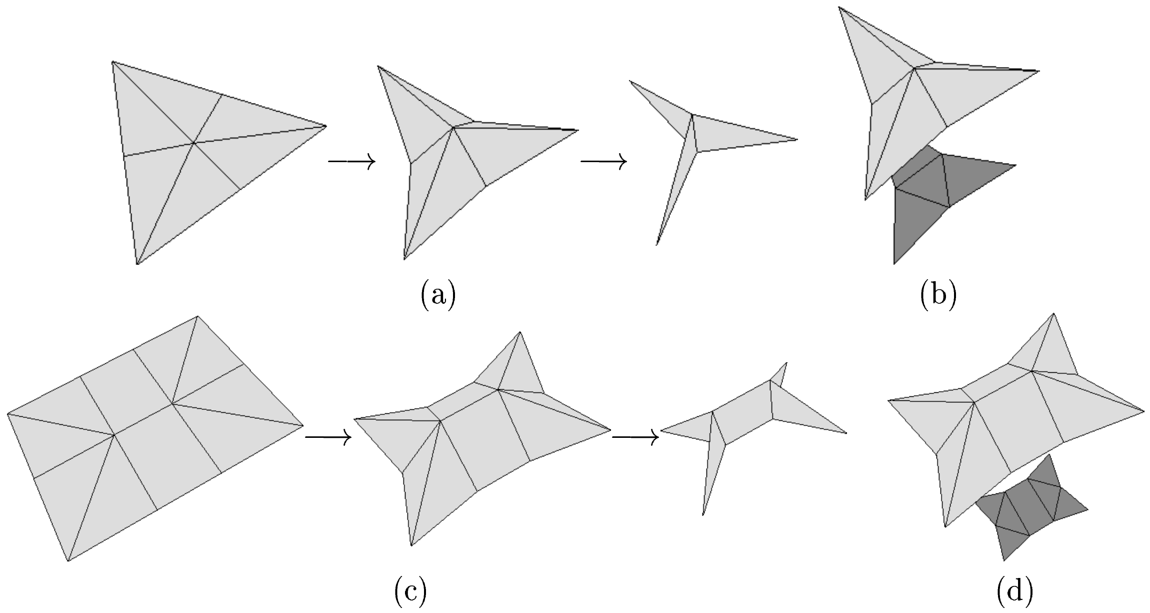

Figure 1: *Unfolded, partially folded, and completely folded extreme bases and the resulting polyhedra for (a–b) a triangle and (c–d) a quadrilateral. To display the entire polyhedron from a single view, we have lowered the bottom piece. Note that these pictures are drawn with a perspective transformation.*

P), and this intersection (called the *shadow*) is a tree. In [13], Lang describes an algorithm to compute the crease pattern of a uniaxial base that has a specified shadow.

In this paper, we restrict our attention to a subset of uniaxial bases, called *extreme bases* [4], that map the boundary of a sheet of paper to a common plane. These bases solve a major part of a fold-and-cut problem (described in Section 2), and have a rigorous foundation. Examples are given in Figure 1.

Because of the coplanar-boundary property of extreme bases, it is important to study coplanar and extreme sets of vertices as the folding occurs. This leads us to investigate the lower convex hull of a partially folded extreme base (Figure 1). By wrapping an elastic membrane around the bottom of a partial folding (that is, forming the lower convex hull), we construct a polyhedron. Note that the topology of the polyhedron is independent of the particular partial folding chosen. We consider this infinite class of polyhedra, which we call *extreme-base polyhedra*, to be part of a more general class of *origami polyhedra*, which come from partial foldings of general uniaxial bases.

Extreme-base polyhedra are the topic of this paper. We use planar graph drawings to help discover properties of these polyhedra. In particular, drawings allow us to demonstrate the recursive structure of the polyhedra, which was not apparent before. For a preview of our drawings, see Figure 6.

We ran into one main problem in our visualizations. When the base is complex, the angles in our drawings become too small. While we could opt for alternative drawing algorithms that improve this, they will not display the graph's recursive structure, which we deem important. To solve this problem, we propose an animated *zoom* facility for interactively traversing the graph to view its details while preserving the recursive structure.

1.1 Related Work in Graph Drawing

Our graphs are similar to *nested-star graphs* [9]. In more detail, if the polygon is non-degenerate, that is, each nonleaf vertex of the straight skeleton has degree three, and we discard "nested triangles" (see Sections 2–3), then our class of graphs is a special case of nested-star graphs. Garg and Tamassia [9] give an algorithm to embed nested-star graphs in the plane with angular resolution $\Theta(1/d^2)$. However, these drawings do not display the recursive structure of our graphs. Indeed, we know of no work attempting to draw recursive structure in any class of graphs.

Our zoom feature is also new, but there are other approaches with similar goals. The idea of *fisheye views* originated from Farrand [5]. It is based on a fisheye camera lens, where objects that are further away from the focus point are viewed as smaller, that is, drawn to a smaller scale. Furnas's paper on *generalized fisheye views* [7] is the foundation for most so-called *emphasis techniques.* His generalization takes into account the importance of objects, in addition to the distance from the focus point. In terms of graphs, Furnas only looked at trees where the importance of a node is proportional to its depth.

Most emphasis techniques are surveyed by Noik [16]. We are aware of two recent papers that were not available when Noik made his survey. The work of Sarkar and Brown [18] can be seen as a generalization of Furnas's ideas to arbitrary graphs; one can focus on a vertex of the graph, and relative importance of vertices is taken into consideration. More recently, Carpendale et al. [19] have explored mapping graphs onto a three-dimensional surface of blended Gaussian curves, thereby allowing multiple focus points and comparison of subgraphs.

1.2 Outline

The rest of this paper is organized as follows. In Section 2, we overview extreme bases, their construction, and the resulting polyhedra. Section 3 presents a linear-time algorithm to find what we call the natural planar embedding. In Section 4, we discuss how we can compensate for small angles by interactively traversing the graph through a "zoom" feature. We conclude in Section 5.

2 Extreme Bases

Extreme bases are uniaxial bases with the property that the boundary of the paper lies on a common plane (the one containing the shadow), while the rest of the paper lies above this plane. The problem of finding an extreme base of a particular shape of paper has been solved when the paper is a triangle and a convex quadrilateral by Husimi and Meguro, respectively [15]; these two "molecules" (shown in Figure 1) are the basis for much origami design. In [4], we generalize these results to arbitrary polygonal paper.

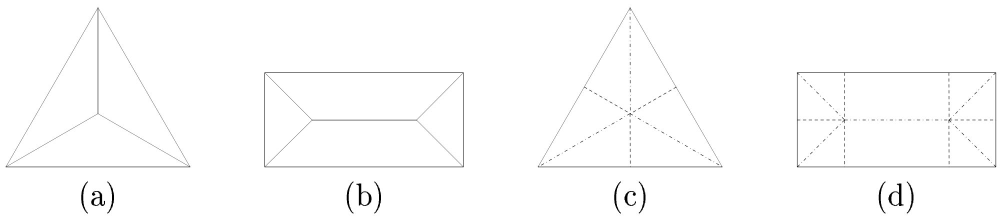

Figure 2: *(a–b) The straight skeleton of a triangle and quadrilateral. (c–d) The crease pattern for an extreme base with triangle- and quadrilateral-shaped paper. Dotted [dot-dashed] lines denote valley [mountain] folds, respectively.*

Our original application for extreme bases is a fold-and-cut problem [8], where one complete straight cut is allowed after arbitrary folding (typically starting from a square piece of paper). Using this single cut, the unfolded pieces can have a variety of shapes. This has been used for a magic trick by Houdini [10] before he became a famous escape artist, and was recreationally studied in detail by magician Loe [14]. Here we consider a special case of the problem, where the goal is to arrange the folding so as to cut out a given polygon $\mathcal{P}$. This means that we want to fold the square paper flat so that the boundary of $\mathcal{P}$ maps to a common line, and nothing else maps to this line; we can then simply cut along this line.

Extreme bases provide a solution to the fold-and-cut problem: we can construct an extreme base for a sheet of paper with shape $\mathcal{P}$, which can then be "flattened" to lie on a plane, where the boundary of $\mathcal{P}$ maps to a line. If $\mathcal{P}$ is convex, we can extend the creases at the vertices of $\mathcal{P}$ (which are angular bisectors) until they reach the end of the square paper. No crossing will occur since the angular bisectors all go "outward" from a convex polygon. We then obtain a flat-foldable crease pattern with $\mathcal{P}$ mapping to a common line, hence solving the fold-and-cut problem for convex $\mathcal{P}$. The solution for nonconvex $\mathcal{P}$ is more complicated but also involves extreme bases.

2.1 Construction

Our construction of an extreme base for a given polygonal sheet of paper $\mathcal{P}$ is based on the *straight skeleton* [1]. For convex $\mathcal{P}$, which is what we will consider in this paper, the straight skeleton is equivalent to the *medial axis* [3, 17]. Conceptually, if we light a prairie fire along the boundary of the polygon, then the medial axis is where the fire meets and burns itself out. With the straight skeleton, fire does not spread in a circular (Euclidean) way. Instead, we uniformly shrink the polygon until it becomes nonsimple; we then shrink each sub-polygon, unless they have zero area. The straight skeleton is simply the trajectories of the vertices of the polygon and sub-polygons from this process.

Examples of the straight skeleton are given in Figure 2(a–b).

The straight skeleton forms a tree of straight line segments with leaves at the vertices of the polygon $\mathcal{P}$, and hence it divides $\mathcal{P}$ into n regions, one for each edge of the polygon. Each nonleaf vertex v has degree three or more, and is adjacent to $\deg(v)$ regions.

The edges of the straight skeleton are the *mountain* folds of the crease pattern for an extreme base of a sheet of paper $\mathcal{P}$. That is, each crease folds higher than the two adjacent regions. There are $\deg(v)$ additional folds incident to each nonleaf vertex v, each perpendicular to an edge of $\mathcal{P}$ and dividing up the corresponding straight-skeleton region. These folds (called *perpendiculars*) are all *valleys*, so the crease goes lower than the two adjacent regions in the crease pattern.

In [4], we prove that these folds form an extreme base. Examples of the crease pattern are given in Figure 2(c–d).

3 Natural Drawing

Before we attempt to draw the graph of an extreme-base polyhedra, we must first define it. Clearly, it includes the crease pattern and the boundary of $\mathcal{P}$. The lower convex hull corresponds to adding edges to this graph. To determine which edges, we need a property about the partially folded extreme base.

Lemma 1 [4] *At any point during the folding of an extreme base, the ends $v_1, \ldots, v_{\deg(v)}$ of the perpendiculars incident to a common straight-skeleton vertex v are coplanar. Furthermore, all other points lie above this plane.*[1]

This means that the polygon with vertices $v_1, \ldots, v_{\deg(v)}$ is a face of the polyhedron. Hence, we connect these vertices in a cycle consistent with the order of the straight-skeleton regions (i.e., polygon edges).

The problem with typical drawings of the polyhedron, such as a top or bottom view, or even a three-dimensional view, is visibility. We want a drawing of the entire graph. Clearly, the wireframe of every polyhedron (without holes) has a planar embedding, and hence the graph is planar. Indeed, it must then have a planar straight-line drawing [6].

The easiest way to construct the geometry of a planar embedding is to use curved edges (Figure 3). This drawing has the property that the vertices of the polygon, straight skeleton, and perpendiculars are in the same positions as the crease pattern. To add the curved edges, first root the straight-skeleton tree at a nonleaf vertex r. Then process a vertex v whose children are all leaves or have been processed already, by cyclicly connecting $v_1, \ldots, v_{\deg(v)}$, staying in the "section" defined by the perpendiculars from v's parent. Since r does not have a parent, it must be processed specially: cyclicly connect the perpendicular ends such that the added cycle becomes the boundary.

One can imagine "stretching" the outside cycle to straighten out the curves (Figure 4), and recursing until we have a straight-line planar drawing. However, this is difficult to describe algorithmically, and leads to an unsatisfactory placement of points without further modification. We shall now give an algorithm to construct what we call the *natural* drawing directly from the straight skeleton. Again we root the straight-skeleton tree at some nonleaf vertex r.

[1]There are certain degenerate situations where some other points lie on this plane, because of unusual coplanarity. For example, if $\mathcal{P}$ is a rectangle, all the ends of the perpendiculars are coplanar in a partial folding. However, this is not the case for almost every convex quadrilateral with topologically the same straight skeleton. Throughout we ignore such degeneracies.

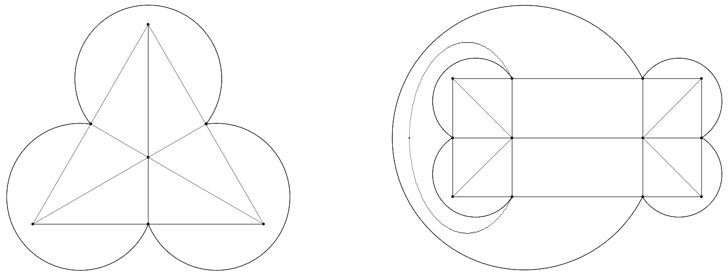

Figure 3: *Planar (curved) drawing of the graph for (a) a triangle and (b) a quadrilateral.*

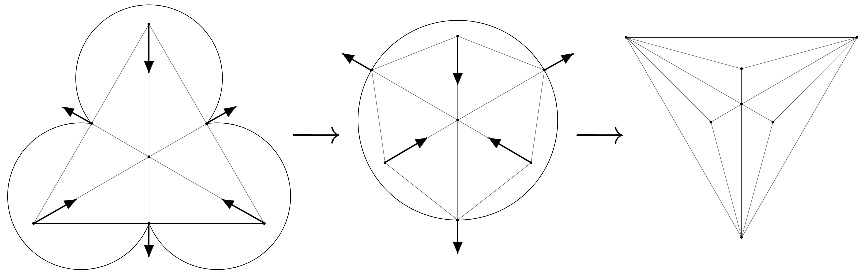

Figure 4: *Stretching the curved drawing of the graph for a triangle into a planar straight-line drawing.*

Algorithm Natural

- Draw a regular $\deg(r)$-gon, filled in with a straight skeleton, resulting in triangles $T_1, \ldots, T_{\deg(r)}$, in clockwise order (Figure 5(b)).
- Mark the obvious edge of each T_i as outside.
- Let $c_1, \ldots, c_{\deg(r)}$ denote the children of r in clockwise order. Then for each $1 \le i \le \deg(r)$, call Recurse (c_i, T_i).

Algorithm Recurse (v, T)

1. If v is a leaf, fill in T with a straight skeleton and return (Figure 5(d)).
2. Add a triangle N nested inside T, and join the respective corners of the two triangles.
3. Mark as outside the edge $\{v_1, v_2\}$ of N corresponding to T's outside edge $\{w_1, w_2\}$.
4. Add a connecting path of length $\deg(v)$ from v_1 to v_2, thereby adding $\deg(v) - 2$ (> 0) new vertices $n_1, \ldots, n_{\deg(v)-2}$ (their positions are discussed in Section 3.1). Let v denote the other vertex of N. Add an edge from v to n_i for $1 \le i \le \deg(v) - 2$, resulting in triangles $T_1, \ldots, T_{\deg(v)-1}$ in clockwise order (Figure 5(f)).
5. Mark as outside the edge of T_i adjacent to the $\deg(v)$-gon inside N that includes $\{v_1, v_2\}$.

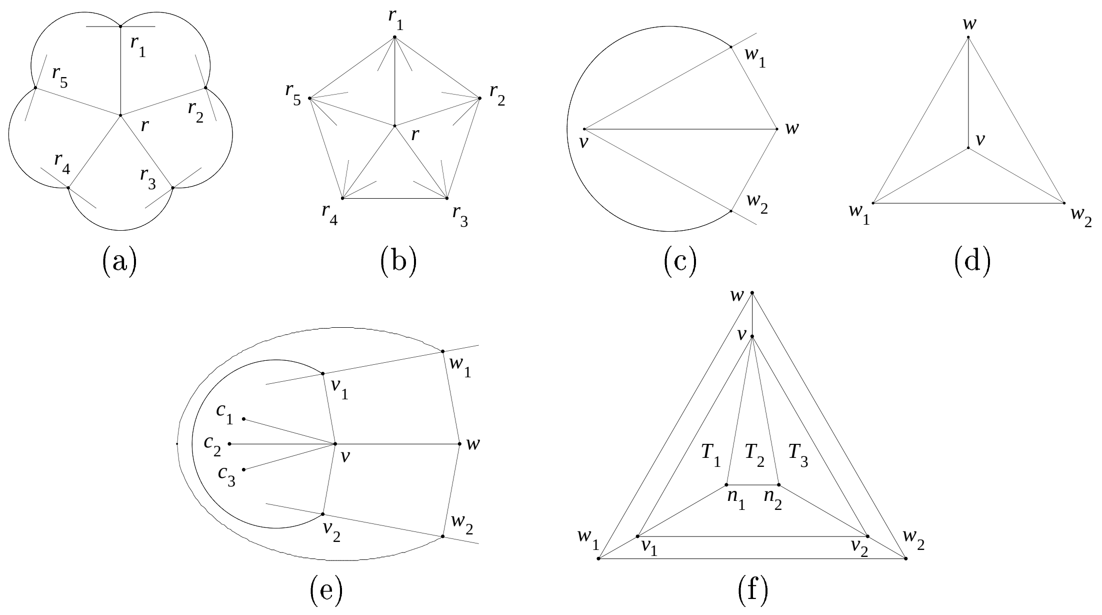

Figure 5: *Illustrative proof of Theorem 1. (a) Curved drawing around the straight-skeleton root r, where* $\deg(r) = 5$. *We draw a portion of the polygon to show the perpendiculars. (b) Corresponding natural drawing. (c–d) Curved and natural drawings around nonroot v with parent w, where* $\deg(v) = 1$ *(leaf). We draw the outside edge on the bottom. (e–f) Similarly for* $\deg(v) = 4$ *(nonleaf).*

6. Let $c_1, \ldots, c_{\deg(r)-1}$ denote the children of v in clockwise order. Then for each $1 \leq i \leq \deg(r) - 1$, call Recurse (c_i, T_i).

Note that if $\deg(v) = 3$, which is normally the case, then Step 4 is equivalent to filling N with a straight skeleton and setting T_1 and T_2 to the two triangles that are not adjacent to $\{v_1, v_2\}$.

In addition to this natural drawing, we have tried moving the vertices to improve angular resolution and area usage. While this can improve the viewability of drawings considerably, it fails to convey the *recursive structure* illustrated in Algorithm Natural. An example of this recursive structure is the lower-right corner of Figure 6. Here, the drawing is essentially several nested straight skeletons of triangles. In general, the drawings consist mainly of nested straight skeletons of regular polygons. We feel this structure is important to display, and hence choose the "natural" drawing.

Theorem 1 *Algorithm Natural is correct and runs in* $O(n)$ *time, where* n *is the number of vertices in the polygon.*

Proof: The algorithm spends a constant amount of time for each end of a perpendicular, of which there are a linear number, and hence the time bound follows.

Figure 5 demonstrates the isomorphism between the curved and natural drawings. An important preserved property is that, for each call to Recurse, T's outside edge corresponds to the curve enclosing v's "section," and the vertex w opposite the outside edge corresponds to v's parent. This follows easily by induction. ■

3.1 Vertex-Placement Details

To simplify the description of Algorithm Recurse, we have omitted the following two details about the exact placement of vertices:

- the amount by which N is nested within T (Step 2), and
- the positions of $n_1, \ldots, n_{\deg(v)-2}$ (Step 4) when $\deg(v) > 3$.

The first question has no one right answer. Let n and t denote corresponding vertices of triangles N and T, and let c denote the centroid (i.e., I-point) of N, which is the same as the centroid of T. To define n, we force it to be on the segment t, c, and force that $|n - t| = \alpha|t - c|$ for some constant $0 < \alpha < 1$. We use $\alpha = 1/4$, a value obtained empirically, but it may be varied to suit the viewer.

Similarly, we force n_1 and $n_{\deg(v)-2}$ to be on the angular bisectors v_1, c and v_2, c, respectively, such that $n_1, n_{\deg(v)-2}$ is parallel to v_1, v_2. If $\deg(v) > 4$, the remaining n_i $(2 \leq i \leq \deg(v) - 3)$ are (unequally) spaced along the segment from n_1 to $n_{\deg(v)-2}$. The n_i $(1 \leq i \leq \deg(v) - 2)$ are placed so that $\angle v_1, v, v_2$ is split into $\deg(v) - 1$ equal pieces. Note that this determines, for example, where n_1 is along v_1, c.

Examples of the resulting drawings are given in Figure 6.

4 Zoom

It should be clear from Figure 6 that, once the straight-skeleton tree becomes sufficiently deep, the natural drawing will not be entirely visible. If the straight skeleton has depth d and only has vertices of degree ≤ 3, then the angular resolution of the natural drawing is $O(2^{-d})$, which is very poor.

To solve this problem, we propose an interactive *zoom* feature. The idea is that the user can navigate through portions of the graph to capture its overall structure.

The initial view is the natural one, truncated after a certain depth to increase rendering time and avoid cluttering. Triangles with contents that could not be shown because of truncation are filled with gray. The user can select a "top-level" nonempty region, that is, a triangle T with depth immediately below the current view. This will cause T to reshape to an equilateral triangle that fills the screen, with the outside edge at the bottom. The portion of the graph (theoretically) visible is hence the subtree of the straight skeleton rooted at the vertex corresponding to T.

Zoom operations display animations to transform from the current view to the desired one, so that the user can keep track of what happens. In the first zoom operation, T is a triangular section of a regular $\deg(r)$-gon P, which say has one edge e at the bottom of the screen, corresponding to triangle T' at the same depth as T. If $T \neq T'$, we rotate P so that T is in the position that T' started in. We then stretch T to a screen-filling equilateral by gradually moving the vertices to the obvious positions at a uniform rate. Note that two of the vertices will only move horizontal, and the other only moves vertical, which we believe is a simple-enough movement for the user to understand. Finally, we hide the rest of P.

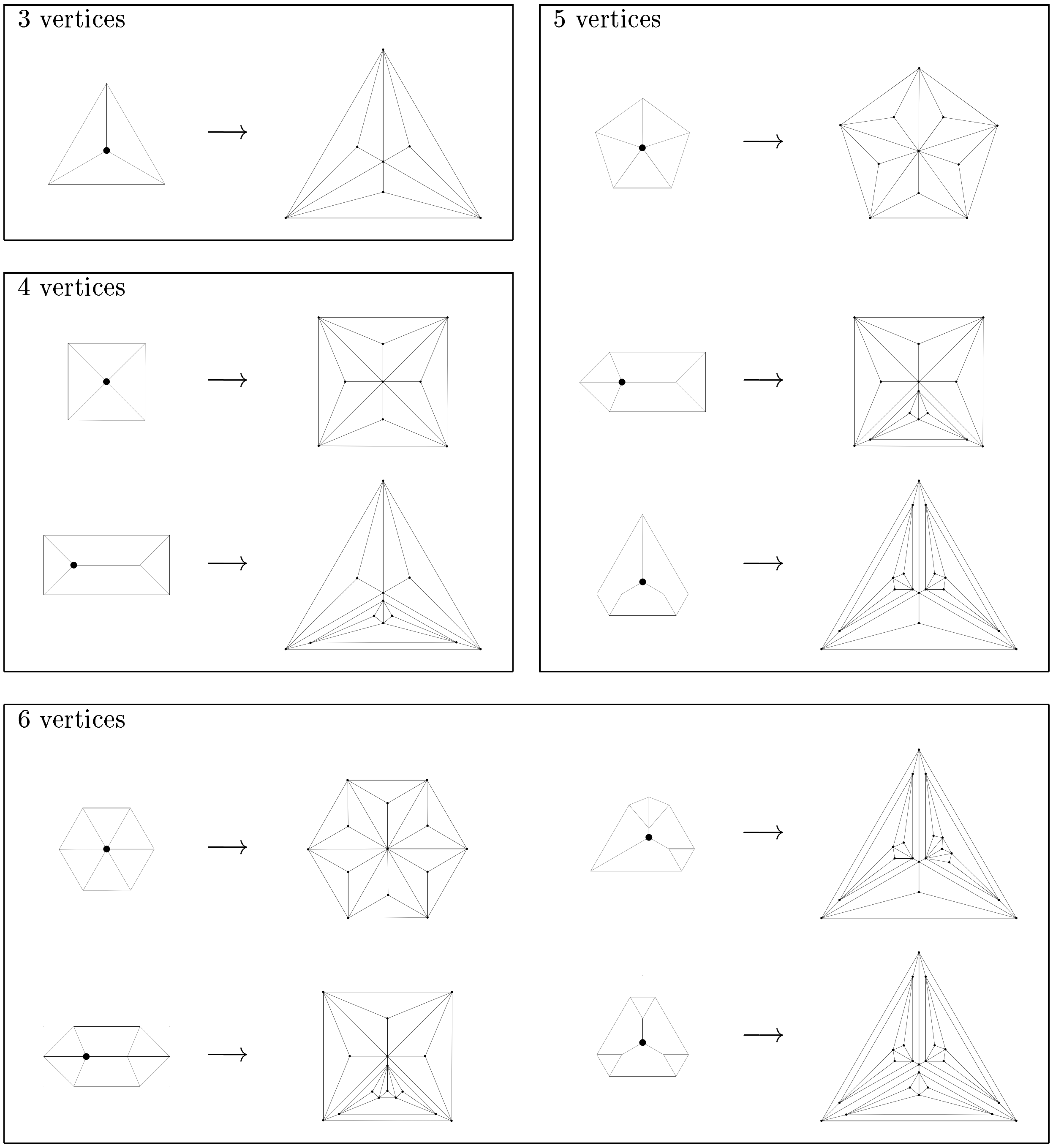

Figure 6: *All possible convex polygons, in terms of the topology of the straight skeleton, with 3–6 vertices, and the corresponding natural drawings of the origami polyhedra. The chosen root of each straight skeleton is marked with a filled circle.*

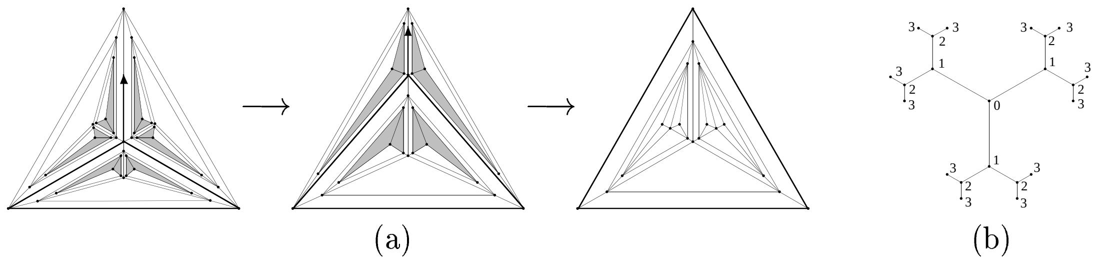

Figure 7: *(a) Three frames of the zooming animation (from beginning to end) when the user selects the bottom top-level triangle. (b) Corresponding tree view before the zoom. As an example, nodes are labeled with their depth.*

See Figure 7(a) for an example of this kind of zooming.

Suppose now that the user selects a triangle T that is more than one level deeper than the root r. We first gradually scale the view so that the containing triangle N, which is nested within the current screen-filling triangle T', becomes screen filling. The top vertex of T is the top vertex of N, and hence is in the correct position. The other two vertices of T are moved uniformly to where the corresponding vertices of N are currently. Note that only one vertex has to move if T is leftmost or rightmost, which is always true for nondegenerate polygons.

Again, we consider this movement to be simple enough for the user to understand. We describe an important property of zooming as follows:

Proposition 1 *Given any view produced by repeated zooming (if any) from the initial view, the user can predetermine the result of a future zoom operation. The user can also determine which boundary edges are outside.*

Justification: The first property is not obvious because the effect of zooming depends on the depth of the current view. However, this dependence is fairly simple and can be determined visually from the current view. First, assume that the current screen-filling triangle T does not correspond to a leaf in the tree (this can be easily determined since the view would merely consist of T and its straight skeleton). In this case, the view is deeper than the initial view if and only if T has a nested triangle N immediately within it. Hence, whether such an N exists determines what type of zoom will occur.

Similarly, if no such N exists, that is, we are at zero depth, then all boundary edges are outside. Otherwise, the bottom of T is the unique outside boundary edge. ■

The user can also unzoom to the previous depth. This essentially corresponds to an "undo" operation, that is, the reverse of the previous animation. Note that the proof of Proposition 1 also shows that the user will recognize when s/he has returned to the initial view and further unzooming is impossible; the outermost triangle will be filled in (Figure 5(d)), not nested (Figure 5(f)).

Proposition 1 argues that zooming as we have described so far is a useful navigation facility. However, it is possible that the stretching of triangles causes the user to get disoriented and "lost." To remedy this, we display a parallel *tree view* of the graph that does not change its shape. It is topologically the straight skeleton being traversed, but is drawn so that the angular resolution is optimal (Figure 7(b)). That is, if d is the degree of a vertex v, then the angles at v are $2\pi/d$. Since we disallow crossings, this requires a lot of area, but we do not expect to draw the entire tree in one view. Instead, we want to show a few depths above and below the current one.

The user can select on tree vertices as an alternative way to specify how to zoom or unzoom, which permits multiple operations in a single input. With either method, the tree view is animated in parallel with the graph view. Basically, the center of the tree view gradually moves to the vertex corresponding to the selected triangle. The tree is simultaneously scaled so that the new vertex has the right scale, and hence the same number of depths above and below are visible.

We find that the tree view aids the user from getting lost. It also allows us to communicate additional information. For example, we can label each vertex of the tree with its depth and height. We can also color vertices to indicate whether they have been visited before, which would be useful if the user performed something like a depth-first traversal.

Note that truncation of the display allows an infinite graph, which is easy to define by using an infinite tree instead of the straight skeleton of a polygon. A simple rule such as each vertex having degree three defines a self-similar (or *fractal*) drawing. The (lazy) generation of the infinite tree can be randomized to produce more interesting results.

5 Conclusion

In this paper, we have described an experiment in the visualization of extreme-base polyhedra. As a result, we have a much better understanding of the underlying extreme bases. In the future, we hope that this investigation leads to new discoveries in origami mathematics. In addition, the zoom feature that we have presented may well apply to drawings of other graphs.

One possibility for future work is to look at the extreme bases of nonconvex polygons. This situation is much more complex [4], and the resulting graphs may be worth studying for further insight into extreme bases.

Acknowledgment

We would like to thank Anna Lubiw for valuable comments on this paper.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC).

References

[1] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995.

[2] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175–183, Atlanta, January 1996.

[3] Harry Blum. A transformation for extracting new descriptors of shape. In W. Whaten-Dunn, editor, *Proceedings of the Symposium on Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[4] Erik D. Demaine and Martin L. Demaine. Computing extreme origami bases. Technical Report CS-97-22, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1997.

[5] W. A. Farrand. *Information Display in Interactive Design.* PhD thesis, Department of Engineering, University of California, Los Angeles, 1973.

[6] I. Fáry. On straight line representations of planar graphs. *Acta Scientiarum Mathematicarum (Szeged)*, 11:229–233, 1948.

[7] George W. Furnas. Generalized fisheye views. In Marilyn Mantei and Peter Orbeton, editors, *Human Factors in Computing Systems — III: Proceedings of the CHI'86 Conference*, pages 16–23, Boston, MA, April 1986. ACM Press.

[8] Martin Gardner. Paper cutting. In *New Mathematical Diversions (Revised Edition)*, Spectrum Series, chapter 5, pages 58–69. The Mathematical Association of America, Washington, D.C., 1995.

[9] Ashim Garg and Roberto Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In Jan van Leeuwen, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms (ESA'94)*, volume 855 of *Lecture Notes in Computer Science*, pages 12–23, Utrecht, the Netherlands, September 1994. Springer-Verlag.

[10] Harry Houdini. *Paper Magic*, pages 176–177. E. P. Dutton & Company, 1922.

[11] Thomas Hull. On the mathematics of flat origamis. *Congressum Numerantium*, 100:215–224, 1994.

[12] Thomas Hull. Personal communication, January 1997.

[13] Robert J. Lang. A computational algorithm for origami design. In *Proceedings of the 12th Annual Symposium on Computational Geometry*, pages 98–105, Philadelphia, PA, May 1996.

[14] Gerald M. Loe. *Paper Capers.* Magic, Inc., Chicago, 1955.

[15] Jun Maekawa. Evolution of origami organisms. *Symmetry: Culture and Science*, 5(2):167–177, 1994.

[16] Emanuel G. Noik. A survey of presentation emphasis techniques for visualizing graphs. In Wayne A. Davis and Barry Joe, editors, *Proceedings of Graphics Interface '94*, pages 225–233, Banff, Alberta, May 1994.

[17] Joseph O'Rourke. *Computational Geometry in C*, pages 193–195. Cambridge University Press, 1994.

[18] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–83, December 1994.

[19] M. Sheelagh, T. Carpendale, David J. Cowperthwaite, F. David Fracchia, and Thomas Shermer. Graph folding: Extending detail and context viewing into a tool for subgraph comparisons. In Franz J. Brandenburg, editor, *Proceedings of the Symposium on Graph Drawing (GD'95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 127–139, Passau, Germany, September 1995. Springer.