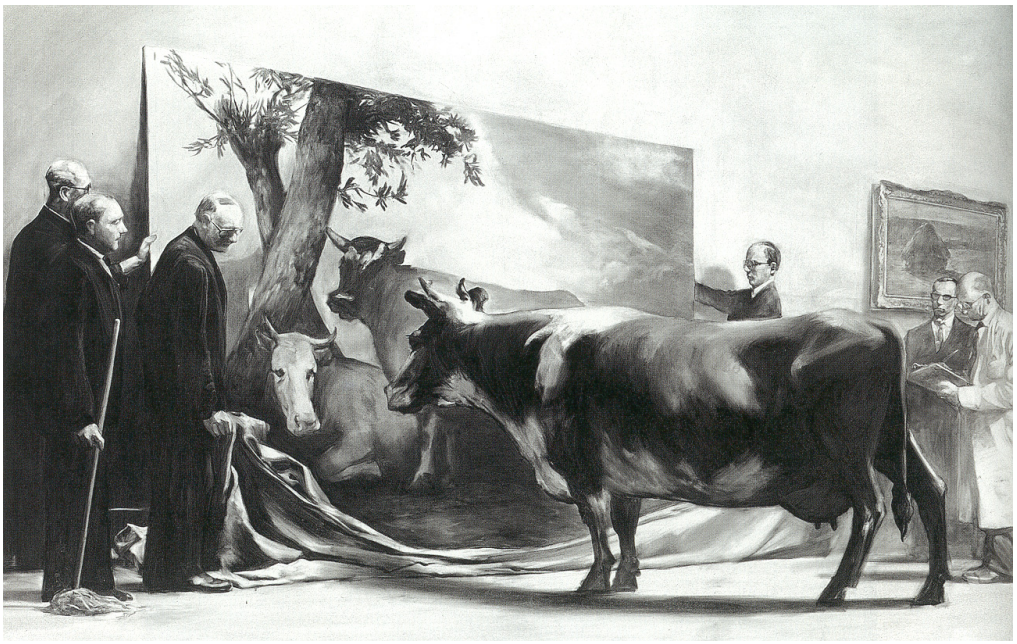
Unfolding Polyhedra

Phillips Alexander Benton
September 7, 2008



This dissertation is submitted for the degree of Doctor of Philosophy.

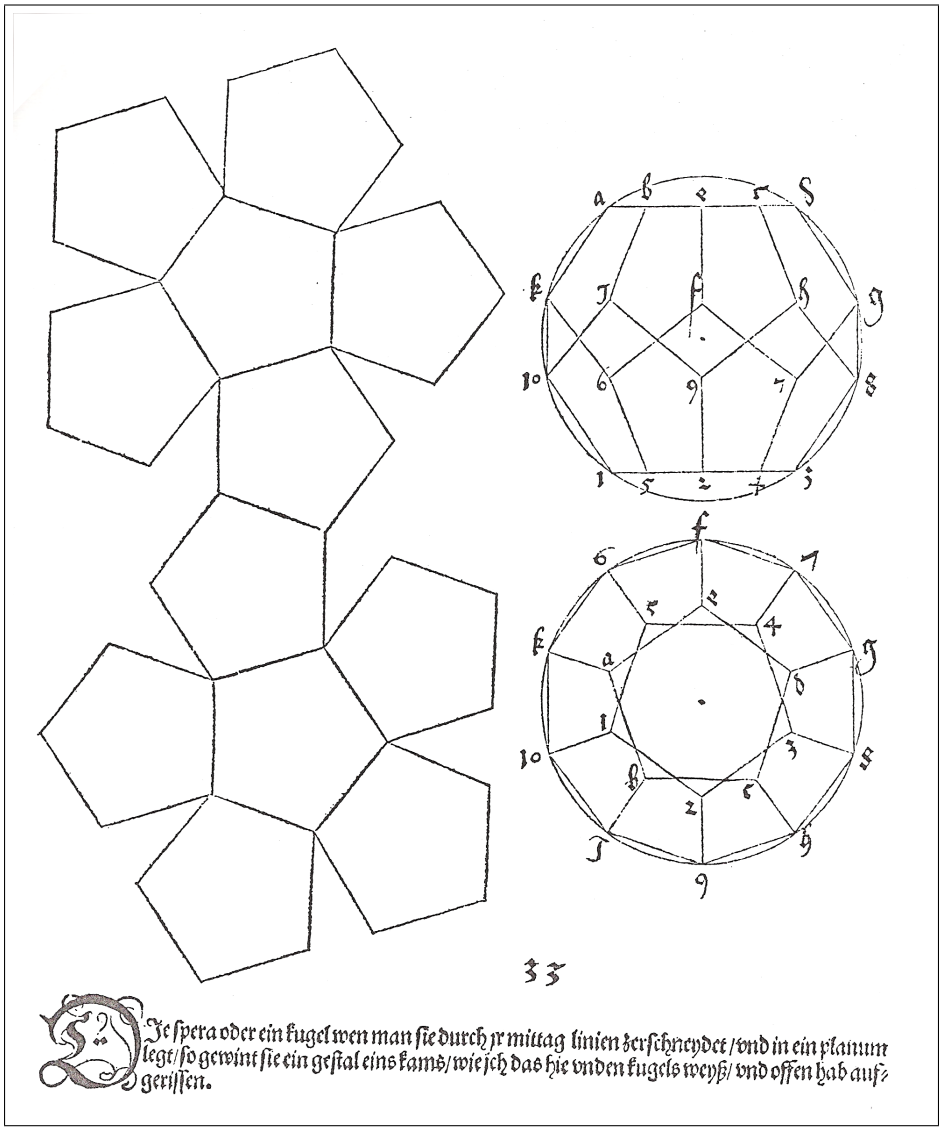
Abstract

Title: Unfolding Polyhedra
Author: Phillips Alexander Benton

It is a common conjecture that all convex polyhedra must be edge-unfoldable but to date a valid proof of this has escaped discovery. This dissertation presents several new directions in the quest for the proof. Also discussed is a method which may lead to a counterexample to the conjecture through the construction of 'hard to unfold' polyhedra.

Algorithmic solutions are discussed for the task of determining the specific set of edges which must be cut in order that an unfolding not self-intersect. A series of *Unfolder* algorithms are explored and compared, in terms of both algorithmic design and empirical performance on test data.

No surface of uniformly negative internal curvature with fewer than two border curves is unfoldable. The *coolinoids* are a class of non-convex polyhedra having exactly two border curves and negative curvature at every internal vertex, which may be constructed so as to be unfoldable without overlap. The fascinating interaction between construction and overlap in coolinoids is modeled and explored.



Albrecht Dürer's net of a dodecahedron (Dür25, Fig.33)

Preface

Almost 500 years ago...

In 1525 at the heart of the European Renaissance, in a time when Mathematics and Art were much closer bedfellows than they are today, the artist Albrecht Dürer published *Underweysun der Messung* ("*The Painter's Manual*"). Dürer wanted to teach his fellow artists and architects how they could re-create variations on Euclid's *corpa regularia*, the Platonic Solids. Dürer provided a new sort of diagram: he showed each solid inscribed within a sphere, as Euclid had done, and then showed the same solid opened up, unfolded to what he called the 'ground plan' of the shape: what today we call the *net* or *unfolding*. This marked the first recorded use of unfoldings in history.

Since Dürer's day unfolded nets have spread far and wide. The *Developable Surface* series of sculptures by Antoine Pevsner (1884–1962) was based on ruled surfaces which could be unfolded into the plane. Many commercial products available today are assembled from folded forms in steel or plastic; automated steel-bending and assembly is an active field of research (GBKK98). Even motorcycle enthusiasts are venturing into unfolding: papercraft models are becoming ever more popular through the internet, to the point where Yamaha Motors now offers online blueprints to let aficionados cut out, fold up and assemble copies of their favorite Yamaha bikes in the comfort of home (Yam05).

Dürer may never have planned on papercraft motorcycles but he did, inadvertently, raise some very interesting questions. *The Painter's Manual* is replete with unfolded nets of platonic solids and their truncations. How did Dürer find his unfolded nets in 1525? Was it simply luck that he chose unfoldings which did not overlap, or was there method to his choice? Beneath his innocuous manual of instruction lurks fascinating questions.

This dissertation is the result of the author's own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

Cover image: "The Innocent Eye Test", Mark Tansey, 1981.
Permanent collection of the Metropolitan Museum of Art, New York City, NY.
http://www.metmuseum.org.

Contents

Chapter 1

Introduction

1.1 Thesis

This dissertation examines two long-unanswered questions:

- Can every convex polyhedron be *edge-unfolded*; that is, can every convex polyhedron be cut along some of its edges and unfolded flat into a simple polygon which does not overlap itself? And,
- Given an arbitrary polyhedron, convex or non-convex, how can an unfolding of the polyhedron be efficiently found; or, if none exists, how quickly can this fact be determined?

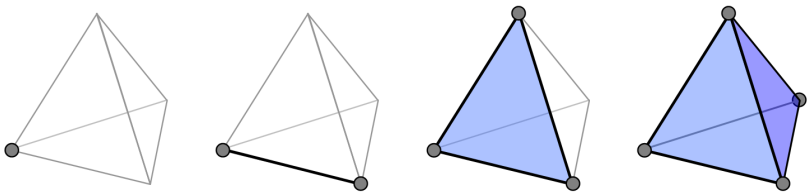
The following new results are presented:

- A series of negative results are presented on the construction of the proof that all convex polyhedra are unfoldable.
- Several positive approaches are proposed, such as a *Euclidean-style proof*, the *alpha-beta rules*, and *error correction*, each of which show promise in the quest for the proof of convex unfoldability.
- A new form of construction which produces 'hard to unfold' convex polyhedra, *polyhedral banding*, is discussed as a possible path toward building a convex polyhedron which is *not* unfoldable.
- A series of algorithms for unfolding convex and non-convex polyhedra are discussed and compared, outlining the strong points and shortcomings of each.
- The *coolinoid*, a surface of uniformly negative internal angle deficit, is explored in detail and an automated method for unfolding a coolinoid is described.

1.2 Terminology

1.2.1 Geometric primitives

Point, Vertex A *point* or *vertex* is a location in $\Re^3$. The term 'point' is commonly used to describe a location in space in its own terms, whereas



(a) $\Re^0$ (b) $\Re^1$ (c) $\Re^2$ (d) $\Re^3$

Figure 1.1: Minimal forms (*simplexes)* in $\Re^k$: (a) Point (b) Edge (c) Triangle (d) Tetrahedron

'vertex' is often used when speaking of a point in the larger context of connectivity with other data. The scalar distance between two points A and B is denoted $|A - B|$[1].

Edge An *edge* is the set of all points in $\Re^3$ which lie on the line segment between two vertices. The edge AB is defined by endpoints A and B as the set of all points P such that $P = A + t(B - A),\ t \in [0 \dots 1]$.

Polygon A *polygon* is a coplanar non-empty set of points in $\Re^3$ bounded by a closed, connected set of edges, in which edges meet only at their vertices and exactly two edges meet at every vertex. If a polygon is *convex* then for any two edges A and B in the polygon, all points in B which are not in A lie to the same side of the line containing A.

The angle at which two edges AV and BV meet at the vertex V of a polygon F is called the *incident angle* of F at V. The incident angle is the area of the intersection of the polygon with a circle of small radius r centered at V, divided by $2/r^2$. If F is convex, the incident angle $\alpha(F, V)$ of F at V is more easily calculated as

$$\alpha(F, V) = \cos^{-1}\left(\frac{(A - V) \cdot (B - V)}{|A - V||B - V|}\right) \tag{1.1}$$

Polyhedron A *polyhedron*[2] (pl: *polyhedra*) is a connected set of polygons which meet only at their edges with no more than two polygons meeting at

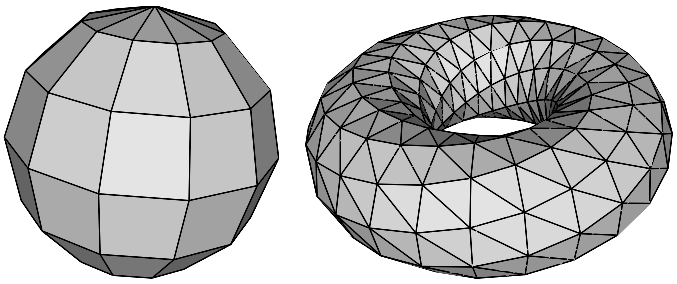
[1] This should not be confused with the notation for the number of elements in a set S, which will be denoted by $\|S\|$.

[2] Given that it is natural for humans to want to describe a discrete 'shell of a thing' in unambiguous mathematical terms, it is unsurprising that a number of definitions of polyhedra have arisen over the years. These definitions broadly break down into two categories: those which describe polyhedra as volumes, whose surfaces have traits akin to those given here; and those which describe polyhedra as surfaces, often assembled from lesser elements, which follow similar rules.

For example, in his 1932 *Einfachste Grundbegriffe der Topologie* ('*Elementary Concepts of Topology*') (Ale61, p.6), Paul Alexandroff elegantly defines an *r-dimensional polyhedron* as

> ... a point-set of $\Re^n$ which can be decomposed into r-dimensional simplexes in such a way that two simplexes of this decomposition either have no points in common or have a common face (of arbitrary dimension) as their intersection.

... which is to say, he defines a polyhedron as a connected set of tetrahedra. Alexandroff describes a body which is implicitly *solid* (because every non-degenerate tetrahedron has nonzero volume) and is implicitly *without border* because his tetrahedra, defined by 3-simplexes, are



(a) (b)

Figure 1.2: Polyhedra: (a) A non-simplicial genus-zero polyhedron (b) A simplicial genus-one polyhedron

a single edge. A polyhedron composed entirely of triangles is called a *simplicial* polyhedron; a polyhedron in which every edge is shared by exactly two polygons is said to be *closed* or *without border* (sometimes written *without boundary*.) An individual polygon F which is part of a polyhedron P is often called a *face* of P. In fields such as computer-aided geometric design the alternative term *facet* may be used.

Other key terms in the field of polyhedra include:

Convex Polyhedra If a set of infinite half-planes have a finite non-empty intersection, then the surface of their intersection is a convex polyhedron. If a polyhedron is *convex* then for any two faces A and B in the polyhedron, all points in B which are not in A lie to the same side of the plane containing A. (Pin07)

A polyhedron with border which is a subset of a convex polyhedron is called a *convex cap*.

Genus The *genus* g of a polyhedron is

---

without border.

Imre Lakatos considers a number of definitions of polyhedra in his 1976 *Proofs and Refutations* (Lak76). These definitions range from the solid body style,

> A polyhedron is a solid whose surface consists of polygonal faces (Lak76, p.14)

to the surface-oriented definition,

> [A polyhedron is] a system of polygons arranged in such a way that (1) exactly two polygons meet at every edge and (2) it is possible to get from the inside of any polygon to the inside of any other polygon by a route which never crosses any edge at a vertex (Lak76, p.15)

which describes a polyhedron in terms of surface only, placing no constraints on an interior volume.

In the field of unfolding, interest lies always with the surface of the form, and so the author chooses to follow Lakatos' second camp of thought but chooses a slightly more lax definition of a polyhedron. Specifically, by relaxing condition (1) in Lakatos' definition to '*at most* two polygons meet at every edge', the author allows polyhedra with border. This is an essential choice for certain results to be presented. It would, of course, be impossible to speak of polyhedra with border if the conception of a polyhedron were solid.

> ...a topologically invariant property of a surface defined as the largest number of nonintersecting simple closed curves that can be drawn on the surface without separating it. (Wei99)

or, more informally, the number of 'coffee cup handles' the surface possesses. For example, the genus of a torus is one. The genus of every convex polyhedron is zero (Figure 1.2.)

Euler Characteristic A number of results have been found on polyhedra without border. One critical theorem is the Poincaré formula (Wei99):

$$\chi = V - E + F = 2 - 2g \tag{1.2}$$

which states that χ, the *Euler characteristic* of the polyhedron, is equal to the number of vertices in the polyhedron minus the number of edges plus the number of faces. The genus g is equal to $1 - \chi/2$. For every convex polyhedron, $\chi = 2$.

1.2.2 Curves on polyhedra

Polygonal Curve A *polygonal curve* C is a path across a polyhedron, a series of line segments defined by a series of *corners* $C = (c_0, c_1, \ldots, c_{n-1})$ on the surface. Corners may be vertices of the polyhedron, points on edges or points within faces. A curve in which $c_0 = c_{n-1}$ is *closed*.

By convention, polyhedral curves are considered to be directed, with indices increasing counterclockwise. For a closed directed curve, any point on the polyhedron which can be reached by a path from the left-hand side of the curve without crossing the curve is said to lie on the *interior* of the curve.

Interior Angle of a Curve ('Surface Angle') The *interior angle* α_{c_i} of a directed curve at corner c_i is derived in much the same manner as the incident angle of a face at a vertex. (In fact, one might think of the corner and edge segments of the curve as a vertex and two edges, embedded in the surface of the polyhedron.) Thus α_{c_i} is defined to be the total surface area of all points on the polyhedron interior to the curve which are also enclosed within a ball of small radius r centered at c_i, divided by $2/r^2$.

The interior angle at c_i is also sometimes denoted by $\lambda(c_i)$, the *left interior angle* of c_i. This is complemented by $\rho(c_i)$, the *right interior angle*. ρ is calculated as above but assumes a clockwise ordering of the vertices of C, which exchanges the interior and exterior of the curve; although the term is defined, the reader is cautioned that such an ordering is not commonly used in computational geometry.

λ and ρ are only defined for vertices at which exactly two incident edges have been cut.

Convex Curve A directed polygonal curve on a polyhedron is a *convex curve* iff $0 < \alpha_{c_i} \leq \pi$ for all c_i.

Smooth Geodesic A *geodesic* on a smooth surface is the shortest path on the surface between two points. On a sphere, this identifies the arc joining

two points; for example, the geodesic between two points on the Earth's equator would be the intervening length of the equator itself.

Discrete Geodesic A geodesic path on a polyhedron becomes a *discrete geodesic*, which is a type of polygonal curve. In (AAOS90) a *geodesic on a polyhedron* is defined to be

> [...]a path π on [the polyhedron] that cannot be shortened by a local change at any point in its relative interior. Equivalently, a geodesic on the surface of a convex [polyhedron] is either a subsegment of an edge or a path that (1) does not pass through corners, though may possibly terminate at them, (2) is straight near any point in the interior of a face and (3) is transverse to every edge it meets in such a fashion that it would appear straight if one were to [apply a solid body transformation to each of the two faces so that their normals are coincident.]

1.2.3 Curvature of discrete surfaces

Gaussian Curvature The *Gaussian curvature* of a region of a surface is the ratio between the area of the unit sphere swept out by the normals of that region and the area of the region itself, and the Gaussian curvature of a point is the limit of this ratio as the region tends to zero area. On a discrete surface, however, normals do not vary smoothly; the normal to a face is constant on the face, and at edges and vertices the normal is–strictly speaking–undefined. Thus normals change instantaneously (as one's point of view travels across an edge from one face to another) or not at all (as one's point of view travels within a face.) The Gaussian curvature of the surface of any polyhedral mesh is therefore zero everywhere except at the vertices, where it is infinite. The total curvature of a discrete surface remains well-defined but to find a useful measure of curvature at an individual vertex, a better solution is needed.

Angle Deficit A number of approaches have been devised for determining the discrete curvature of a surface at a vertex, and Meyer et al. provide a very accessible description of the Gauss-Bonnet scheme in (MDSB02). A simplified form of the Gauss-Bonnet scheme is the *angle deficit* method, well-described by Van Loon in (Loo94), p.5. The angle deficit $AD(V)$ of a vertex V is defined to be:

$$AD(V) = 2\pi - \sum \alpha(f, V) \tag{1.3}$$

Angle Deficit of a Polyhedron The *total angle deficit of a polyhedron* P is the sum of the angle deficits of every vertex of the surface, denoted by Δ. Descartes' theorem of Total Angle Deficit states that (Wei99)

$$\Delta = \sum_{\forall v \in P} AD(v) = 2\pi\chi \tag{1.4}$$

and so the total angle deficit Δ of a surface with genus g is equal to $4\pi\ (1 - g)$. For every convex polyhedron $\Delta = 4\pi$.

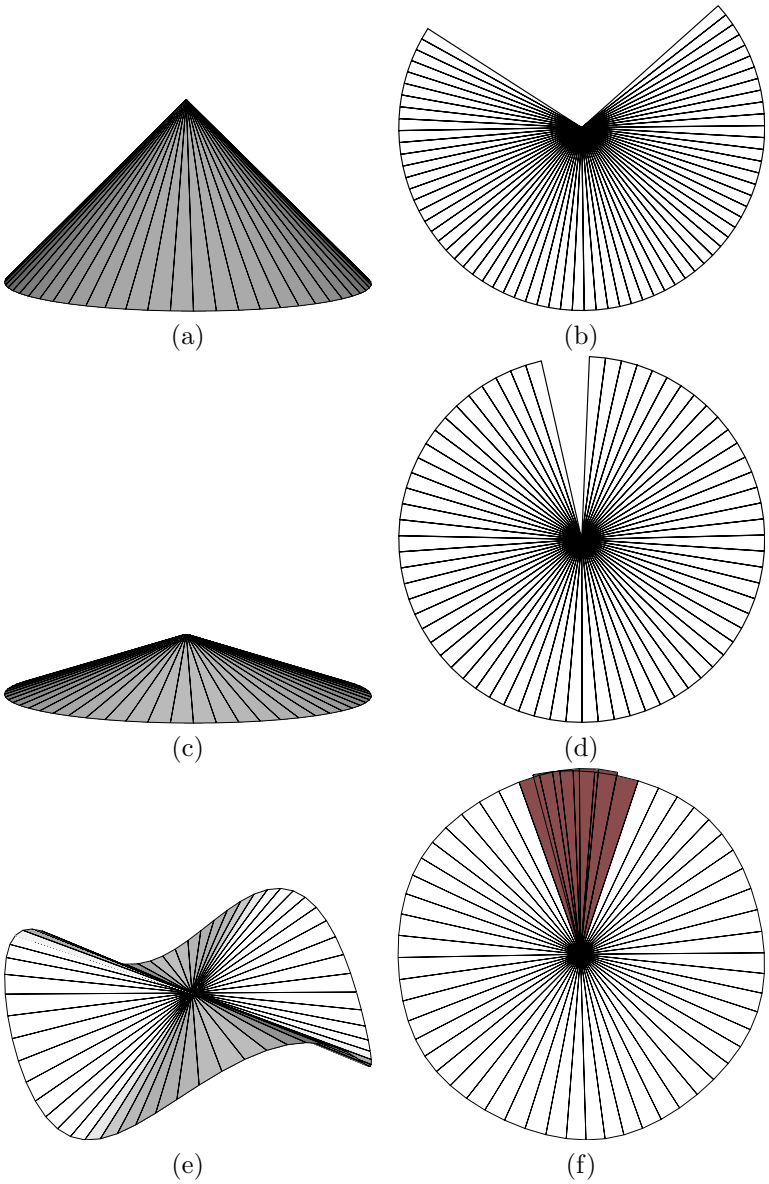


Figure 1.3: (a, b) A vertex with high positive angle deficit; (c, d) A vertex with low positive angle deficit; (e, f) A vertex with negative angle deficit. Vertices with higher angle deficit have higher curvature and flatten to a wider gap when cut open; vertices with negative angle deficit have negative curvature and overlap when flattened.

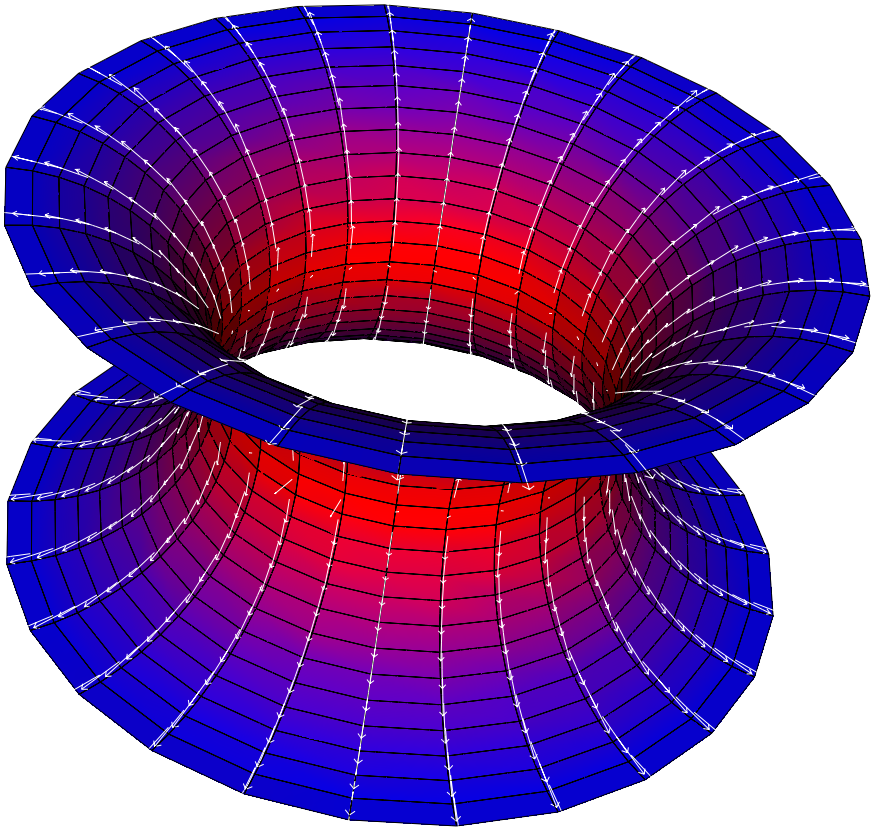


Figure 1.4: The inner half of a torus, shaded by angle deficit. The innermost red band has most negative curvature; arrows indicate the vector field of curvature flow from negative toward zero curvature.

Spherical Vertices A vertex with positive angle deficit is sometimes called a *spherical vertex*. If one were to stand atop a spherical vertex, aligning oneself with the average of the normals of the surrounding faces, then all edges emanating from the vertex would either rise up or drop away, as though one stood atop a mountain or at the bottom of a valley.

Hyperbolic Vertices A vertex with negative angle deficit is sometimes called a *hyperbolic* or *saddle vertex*. If one were to stand atop a hyperbolic vertex, aligning oneself with the average of the normals of the surrounding faces, then some of the edges emanating from the vertex would seem to rise up while other descended, as though one were crossing a pass in a mountain range; slopes ascending to either side but dropping away ahead and behind. More complicated hyperbolic vertices, those with more than the minimal number of two descending and two ascending regions of edges, are called *monkey saddles*.

1.2.4 Graphs and trees

Graph A *graph* is a set of *nodes* linked by *edges*. In a purely abstract graph a node has no inherent value and edges are defined only by the fact that they connect two nodes. Graphs are often used to describe sets of interrelated objects in terms of their connections. (Graph edges should not be confused with polygon edges!)

Valency The *valency* of a node in a graph is the number of edges connecting that node to others. A node of valency one is a *leaf node* of the graph. A node of valency greater than two is a *branch node* of the graph. A branch node of valency 3 is sometimes called a *Y-fork* in the graph.

Path A *path* is a connected, ordered sequence of edges in a graph. Each edge in the path shares one node with the previous edge and one with the next, visiting each node in turn.

Connectivity A graph in which, for every node, there exists a path linking that node to every other node, is *connected*; a graph where this is not the case is called *unconnected*. A graph in which at least k edges must be removed before the graph becomes unconnected is called k-connected; for example, the graph of the connectivity of the vertices of a cube is 3-connected, because a cube has three edges visiting each of its vertices.

Cycles A graph in which there exists no node which can be reached by two distinct (non-overlapping) paths from any other node, is said to be *acyclic*. Conversely if there are two nodes in the graph for which there are multiple distinct (non-overlapping) paths then those nodes are on a *loop* or *cycle*.

Shortest Path The *shortest path* between two nodes is the path which traverses the least number of edges. The concept of shortest path may be improved by assigning spatial locations to graph nodes; then the shortest path can be the path whose total linear length is minimized along the edges in space.

That said, recall that graphs have no implicit spatial configuration. Thus the idea of a shortest path may be extended further to 'weighting' edges, by linear distance or by any other scalar weighting function. Then the shortest path would be the path which minimizes the total 'weight' of the series of edges traversed.

Spanning Graph A *spanning graph* visits every node exactly once. By definition a spanning graph cannot contain a loop, for this would mean visiting the node twice. No edge may be removed from a spanning graph without disconnecting the graph. No edge may be added to a spanning graph without creating a loop. If a graph has n nodes then any spanning of that graph will have exactly $n - 1$ edges.

Hamiltonian Path A connected graph which has no node of degree > 2 is a *Hamiltonian path*.

Planar Graph A *planar* graph may be embedded in $\Re 2$ without crossing its edges. Informally, this means that the graph can be drawn on paper with edges meeting only at the nodes.

Directed Edges Graphs may be *directed* or *undirected*. In a directed graph, edges are oriented and indicate only a one-way relationship between the two nodes.

Tree A *tree* is a connected acyclic graph with orientation. A tree has a distinct *root node*, which is said to be at the 'top' of the tree. From any node of the tree, the *parent* of the node is 'above' the node (with no node above the root) and the *children* of the node are 'below' the node (with no node below the leaves.) Note that in graph theory, unlike reality, trees grow down with the root at the top and leaves at the bottom.

Spanning Tree A *spanning tree* of a graph is an acyclic, connected subset of the edges of the graph. The root node of a spanning tree may be left undefined; many uses of spanning trees are interested only in existence, not orientation. For example, a *minimal spanning tree* of a graph is a spanning tree which minimizes some scalar weighting function on the graph.

Binary Tree A *binary tree* is a tree in which every node except the root node has valency 1 or 3; the root node has valency 1 or 2. In a *strictly binary tree* the root node has valency 1 or 3.

1.2.5 Graphs and polyhedra

1-Skeleton The *1-skeleton* of a polyhedron is the graph of the connectivity of the vertices and edges of the polyhedron. For each vertex of the polyhedron there is a node in the 1-skeleton; for each edge connecting two polyhedral vertices, an edge of the 1-skeleton connects the two corresponding nodes.

Lexicographic Order It is impossible to say that one vertex of a polyhedron is 'greater than' or 'less than' another, for points in $\Re^3$ have no inherent ordering. *Lexicographic ordering* is one way to sort points in a consistent ordering which allows the '$<$' and '$>$' operators to apply. In lexicographic ordering, given two points A and B,

$$\begin{aligned}(A < B) = \text{IF } (A_Y \neq B_Y) \text{ THEN } \quad & (A_Y < B_Y) \\ \text{ELSE IF } (A_X \neq B_X) \text{ THEN } & (A_X < B_X) \\ \text{ELSE } & (A_Z < B_Z)\end{aligned}$$

This ordering can then be applied to the vertices of a face f to lexicographically determine the lowest vertex in the polygon (that is, to find the vertex v_i such that $v_i < v_j$ for all other v_j in f). The vertices can then be ordered in some preset manner, such as counterclockwise about the outwards-facing normal to the face, to give an absolute and consistent ordering of the vertices of the face.

By the same token, the faces of a polyhedron may be lexicographically sorted by their centers and thus absolutely ordered.

1.3 The Unfolding Problem

An *edge-unfolding* of a polyhedron is a cutting of the surface along its edges that unfolds the surface to a single, nonoverlapping piece in the plane.

To be more precise in this definition:

> Let U be the tree of faces of the polyhedron P still connected after some set of edges T in P has been removed. Each node of U is a face of P; each edge of U represents an edge of P which was not in T.
>
> Select an arbitrary face r of P, and compute the solid body transform (a transformation which preserves distances and angles) M_r which will rotate r until its normal is collinear with the Y-axis. The choice of r is immaterial to the computation of the unfolding; different values of r will change the unfolding only in position and orientation.
>
> Then, for every face f in U which is adjacent to a face u for which the solid-body unfolding transform M_u has now been calculated, compute for f the transformation m which will rotate f around the edge which it shares with u until the normals of f and u are equal. Record as M_f the concatenated transformation $(m \cdot M_u)$.
>
> The *edge-unfolding of P by T* is the figure which is the union of each of the faces f of P, transformed by M_f.
>
> If there exists some T such that the edge-unfolding of P by T contains no faces which meet at any point other than at edges which they share in U, then P is said to be *edge-unfoldable*.

The term 'edge-unfolding' will be abbreviated hereafter as simply 'unfolding', unless more precision is called for. (Such as will be the case when discussing *general unfoldings* or *vertex unfoldings*, below.) Likewise any surface which can be edge-unfolded without overlap is termed simply 'unfoldable'.

A number of open questions have been put forth in the literature about unfolding. This work addresses two such questions:

- Are all convex polyhedra unfoldable?
- Can an algorithm be constructed which can find an unfolding of a polyhedron (or declare that none such exists) in 'reasonable' time?

At first glance, it is unapparent why these questions have remained open, for they seem to be fairly intuitive. After all, every orange can be peeled; surely, every orange could be peeled even if one were restricted to a convex set of edges on the surface? Yet a mathematically sound proof of the unfoldability of convex polyhedra has continued to elude researchers. The fact that even very simple polyhedra can be constructed for which *not every* unfolding is without overlap, raises the specter that there might exist some polyhedron for which *no* unfolding is without overlap.

The Open Problem, as phrased by Joseph O'Rourke, is

> Does every convex polyhedron have an edge unfolding to a simple, nonoverlapping polygon, i.e., does every convex polyhedron have a net? (DO07)

By the same token, peeling an arbitrary polyhedron would seem to be relatively straightforward. Yet attempts to actually do so according to rigorous

algorithm have proved remarkably difficult to achieve. It is quite simple to decompose an n-faced polyhedron into several smaller pieces, detached from one another, with the intention of gluing them back together when done (even if such a decomposition must produce n separate pieces;) but to unfold the surface to a single connected net of faces, unbroken, is often surprisingly difficult. Simple non-convex polyhedra have been found which have no unfolding at all, and so any algorithm searching for an unfolding must be prepared to fail.

The challenge, as phrased by Komei Fukuda, is that

> While several simple algorithms have been advanced for the reasonably quick edge-unfolding of a convex surface, it appears that no single quick approach is guaranteed to be effective on all surfaces. (Fuk97)

...and though Fukuda's challenge was put forward for convex surfaces, it generalizes well to the broader field of convex and non-convex polyhedra.

1.3.1 Terminology of unfolding

Cut An edge or series of edges broken in the course of unfolding a surface is referred to as a *cut*.

Cut-Graph Cuts join together to form the *cut-graph*, a connected undirected graph of broken edges. There are key restrictions on the number of loops in the cut-graph, and on the number of loops which may be formed by taking the union of the cut-graph with the graph of the boundary edges of the original polyhedron. The cut-graph is a subset of the 1-skeleton of the polyhedron.

Unfolding Tree The dual of the cut-graph is the *unfolding tree*, the undirected, connected, acyclic and planar graph of the connectivity which remains between faces after all cuts have been made. The unfolding tree is a subset of the graph of connectivity of faces of the original source mesh.

Overlap, Self-Intersection, Collision Two faces which, when unfolded, share points in the unfolding plane other than their edges or vertices, are said to *overlap* or *collide*. The figure formed by the unfolding is then said to *self-intersect*.

Net, Unfolding The figure formed by the unfolding of a polyhedron is the *net* or *unfolding* of the polyhedron. A *valid net* is a net which does not self-intersect.

Partial Unfolding A *partial unfolding* is a connected subset of the unfolded faces of a polyhedron. A *valid partial unfolding* does not self-intersect.

Development A polyhedron which can be unfolded is called *developable*[3], and the unfolded net is the *development* of the surface.

[3]This usage is common in the literature but it is a slight mischaracterization: to refer to a surface as developable stems from the original mathematical definition of a *developable surface* as a surface with zero Gaussian curvature, which can be 'developed' into the plane without stretching or tearing. Examples of developable surfaces include cylinders, which unroll into a plane; cones, which unroll to an arc section; and other, more complex forms, including subsections of other developable surfaces. A sphere–or a polygonal representation

Left and Right Development Given a cut-graph subset without branch, the *left development* of the cut-graph identifies the left-hand set of edges to which that curve unfolds; the *right development* describes the right. (DO07), p. 376.

1.4 Prior Work

Dürer's Renaissance text is the first recorded use of unfolded nets in publication, but it was not until Shepard's 1975 *Convex polytopes with convex nets* (She75) that interest in unfolding began to take a visible role in contemporary research. Since then, a number of interesting avenues have opened in unfolding and related works.

1.4.1 Alternative unfoldings

One key question has been the precise definition of what an 'unfolding' truly is. This dissertation is entirely focused upon edge-unfoldings but there is, as it were, more than one way to skin a cat.

General unfoldings

For example, to limit the cuts of an unfolding to the edges of the polyhedron is an unnecessary restriction: one might also allow cuts across faces, in what is called a *general unfolding*. In a general unfolding the cut-graph is no longer (necessarily) a subset of the 1-skeleton of the mesh; instead it links a set of points on the surface in paths which divide the surface into developable subsections. It has been shown that the *star unfolding* (AO91) and the *source unfolding* (MMP87), methods devised to find the shortest paths between points on the surface, will cut any convex surface and unfold it without overlap.

Vertex unfoldings

Another form of unfolding is the *vertex unfolding*, in which faces are cut apart at multiple edges, retaining connectivity only at their vertices. Vertex unfoldings link faces tip-to-tip, and it is shown in (DEE$^+$02) that every simplicial convex polyhedron can be vertex-unfolded without overlap. However, it remains an open question whether the same holds true for non-simplicial convex polyhedra.

1.4.2 The improbability of unfolding

In (O'R00), Joseph O'Rourke relates results derived from Catherine Schevon's 1989 PhD thesis, *Algorithms for Geodesics on Convex Polytopes*. Schevon found that as n, the number of vertices on sets of randomly-generated convex polyhedra, rose, the percentage of randomly-selected unfoldings of the polyhedra which unfolded to overlap rapidly approached one; that is, as the polyhedra increased in complexity, the odds of their being unfoldable by a randomly-chosen

of a sphere–is *not* developable, for it cannot be flattened without tearing or stretching; but the polyhedron with border which is created by deleting the edges of a cut-graph from that sphere *is* developable, for now the 'tears' have already been torn.

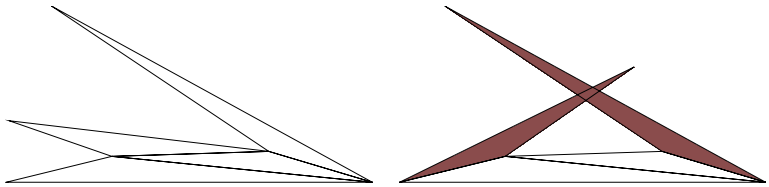


Figure 1.5: Fukuda and Namiki's slim tetrahedron, shown unfolded (a) without and (b) with self-intersection

cut-graph fell to almost zero. And yet, every single polyhedron generated in Schevon's tests did have at least one unfolding which was without overlap.

Schevon's results are mirrored in the statistical analysis of unfolding algorithms conducted for this dissertation, discussed in Section 4.9.

1.4.3 Convex polyhedra with strange unfoldings

Fukuda and Namiki (Fuk97) have shown that even a tetrahedron[4] may be constructed which has an overlapped unfolding (Figure 1.5.) Two of the sixteen distinct possible unfoldings of their slim tetrahedron will overlap.

One result which derives trivially from Fukuda and Namiki's example is that, if one considers the slim tetrahedron to already be cut by a cut-graph which introduces overlap, then it can be known that:

Lemma 1.1 (Fukuda) *There exists a simplicial convex polyhedron with border which cannot be edge-unfolded.*

Proof. By example; see Figure 1.5 and footnote 4 for co-ordinates. □

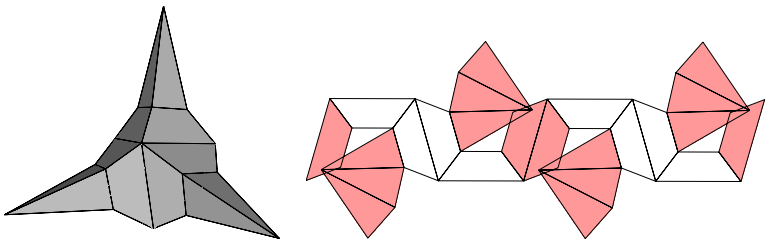
1.4.4 Ununfoldable non-convex polyhedra

In their 1999 *Ununfoldable Polyhedra with Convex Faces* (BDE$^+$03), Bern, Demaine et al. describe a non-convex polyhedron without border which has no edge unfolding. This surface, which they assemble from an undevelopable component which they call a 'witch's hat', can be generally unfolded but provably cannot be edge-unfolded without overlap (Figure 1.6.)

1.4.5 Schlickenrieder and Lucier

In his 1997 Master's Thesis *Nets of Polyhedra* (Sch97), Wolfram Schlickenrieder rigorously explored a series of unfolding methods, trying to find a single unfold-

[4] The co-ordinates of the slim tetrahedron unfolded in Figure 1.5 were derived by the author through experiment and may not represent the exact proportions first used by Fukuda and Namiki. The co-ordinates of the figure shown here are:
[0.0000, 0.0000, 0.0000]
[4.4829, 0.0000, 0.3922]
[6.3000, 0.0000, 0.0000]
[1.8171, 0.4000, 0.1961]
This model, and all others used in this dissertation, is available online. Please refer to Appendix C for details.



(a) (b)

Figure 1.6: Bern, Demaine et al.'s witch's hat assembly, a non-convex surface which cannot be edge-unfolded without overlap. Shown (a) in perspective and (b) unfolded to self-intersection.

ing algorithm which would always unfold every surface. Schlickenrieder grouped his unfolding algorithms as follows:

- Simple unfoldings, such as breadth- and depth-first search
- Shortest-path unfoldings, similar in nature to the general star unfolding
- Directed unfoldings, which used heuristic tree search techniques to choose a cut-graph
- 'Shelled' unfoldings, which trace heuristically-guided paths through the faces of the polyhedron
- Incremental unfoldings, which use heuristic tree search techniques to choose an unfolded net

Despite an extensive investigation, Schlickenrieder failed to find a simple algorithmic solution. While several of his methods–most notably the *shortest-path unfolder* and the *steepest-edge unfolder*–could unfold many of the surfaces described, he was unable to present an algorithm which could unfold *every* convex polyhedron in a single attempt.

In 2006, Brendan Lucier described in his Master's Thesis *Unfolding and Reconstructing Polyhedra* (Luc06, Chap.5) a directed exploration of the conditions which could lead to overlap. Where Schlickenrieder had found specific empirical counterexamples to his own conjectures, Lucier developed several of the counterexamples into mathematically-expressible models whose 'undevelopability' could then be proved. He did so by describing a simple form of collision, the 2-*local* collision, and then showing how to construct almost-flat convex caps which would always unfold to 2-local overlap if unfolded by Schlickenrieder's most viable algorithms. Using these almost-planar figures, Lucier shows how to construct general counterexamples to Shlickenrieder's shortest-path and steepest-edge algorithms.

1.4.6 Angle deficit and cut-graph valence

Konrad Polthier has shown that there is a direct correspondence between the sign of the angle deficit at a vertex and the number of cuts it requires to avoid overlap (Pol03):

- At each spherical vertex there must be at least one cut;
- At each hyperbolic vertex there must be at least two cuts which take off one or more faces during the unfolding.

Vertices of zero angle deficit may be left completely untouched; they can flatten to the plane without cutting a single incident edge.

Although Polthier does not state it explicitly, the following lemma derives trivially from the observations above:

Lemma 1.2 (Polthier) *There can never be a leaf node of the cut-graph at a vertex of negative angle deficit.*

Proof. If a branch of the cut-tree were to terminate at a negative-curvature vertex, only one incident edge would be broken. At least two incident edges must be cut or overlap is unavoidable. □

A second result of Polthier's observation is an interesting note on how $AD(v)$ is calculated. In many applications, angle deficit is rescaled by dividing the sum of the incident face angles by the area of the local Voronoi triangulation, to more closely approximate the Gaussian curvature of a smooth surface passing through the vertex. However, Polthier's observation makes it clear that such a step is unnecessary in the context of unfolding: what is critical at the vertex is the *sign* of its angle deficit, which is unchanged by scalar division. Magnitude beyond sign is of secondary importance.

1.4.7 Loops in the cut-graph

In (Ben07, p.3), the author showed that the cut-graph may not loop, nor may the union of the cut-graph with the boundary graph contain a loop which was not already in the boundary graph, if there is no handle in the topology of the polyhedron. If this condition is not met, the unfolding will become disconnected. The proof is reprinted on page 146 of this work.

From this it is clear that the cut-graph of a genus-zero polyhedron must be a tree.

1.4.8 Closed convex curves

In (OS89), O'Rourke and Schevon show that

Lemma 1.3 (O'Rourke and Schevon) *A closed convex curve [on a convex polyhedron] develops without self-intersection [that is, it unfolds to two connected sets of edges which do not cross each other or themselves.]*

Proof. O'Rourke and Schevon argue that the interior of any closed convex curve C on the convex polyhedron P can be removed, and the convex hull taken of the resulting figure to give a new polyhedron P'. There are no vertices of P' within C and therefore there can be no curvature, and so the interior of C on P' must develop (in the classic sense of the word) to a convex polygon. This establishes that the left-hand development of the curve does not self-intersect. With this construction in place, the remainder of the proof is nicely summarized in (DO07, p.377):

> Let θ_i be the internal angle at the vertex a_i of this planar polygon. A lemma in O'Rourke and Schevon (1989) establishes the plausible claim that $\theta_i \leq \lambda(c_i)$ for all i: cutting away material to form P' can only reduce the angle. Because by assumption we have $\lambda(c_i) \leq \pi$, we may apply Cauchy's arm lemma[5]. Cutting [the right development of C] at any point p and then "opening" the internal angles from θ_i to $\lambda(c_i)$, ensures that the two images of p increase in separation. [...] A further implication [...] is that [the right development of C] does not self-intersect.

□

1.4.9 Classes of unfoldable convex polyhedra

A strong direction in recent research has been the identification of classes of convex polyhedra which can be shown to be unfoldable. Ideally, these classes can be broadened until they encompass all possible convex polyhedra, which would answer the Open Question.

The first such class to be shown to be unfoldable was the platonic solids, as shown by Dürer himself (Dür25) (though this was not his goal.) To date, the following further classes of convex polyhedra have been shown to be unfoldable: *pyramids* (BCO04), *prisms* (DO05), *prismoids*[6] (DO05) and *smooth prismatoids*[7],[8] (BCO04), and *domes*[9] (DO07; BO07).

In (Pin07) Val Pinciu gives an easily intuitive proof of the *volcano unfolding* of the 1-ring of a single face on a convex polyhedron, which forms a convex cap. If the *neighborhood* $N[A]$ is defined to be the face A and the set of faces sharing an edge with A on convex polyhedron P then Pinciu defines the volcano unfolding by cutting all edges of $N[A]$ except the edges which bound A and shows that all volcano unfoldings are free of overlap. Volcano unfoldings are also defined in (DO07), although the authors do not provide a formal proof.

Although not stated explicitly, a direct result of Pinciu's proof is that:

Lemma 1.4 (Pinciu) *All four-faced closed convex polyhedra are unfoldable.*

Proof. Any three faces of a tetrahedron must each have a unique edge in common with the four face. If these shared edges are left uncut while all other edges on the surface are cut, the result is a volcano unfolding and therefore free of overlap. □

1.4.10 Further useful lemmas

Cauchy A useful result often cited in fields such as linkage problems and robotic path-planning is *Cauchy's Arm Lemma*. Essentially, it states that if the angle at a vertex of a convex polygon is increased while all but one of its

[5] see p. 18

[6] *Prismoid*: the convex hull of two equiangular convex polygons, oriented so that corresponding edges are parallel (DO05)

[7] *Prismatoid*: the convex hull of two convex polygons in parallel planes (BCO04)

[8] *Smooth prismatoid*: the convex hull of two C^2-continuous convex curves in parallel planes (BCO04)

[9] *Dome*: a convex polyhedron with a distinguished base polygon B and the property that every nonbase face shares an edge with B (BO07)

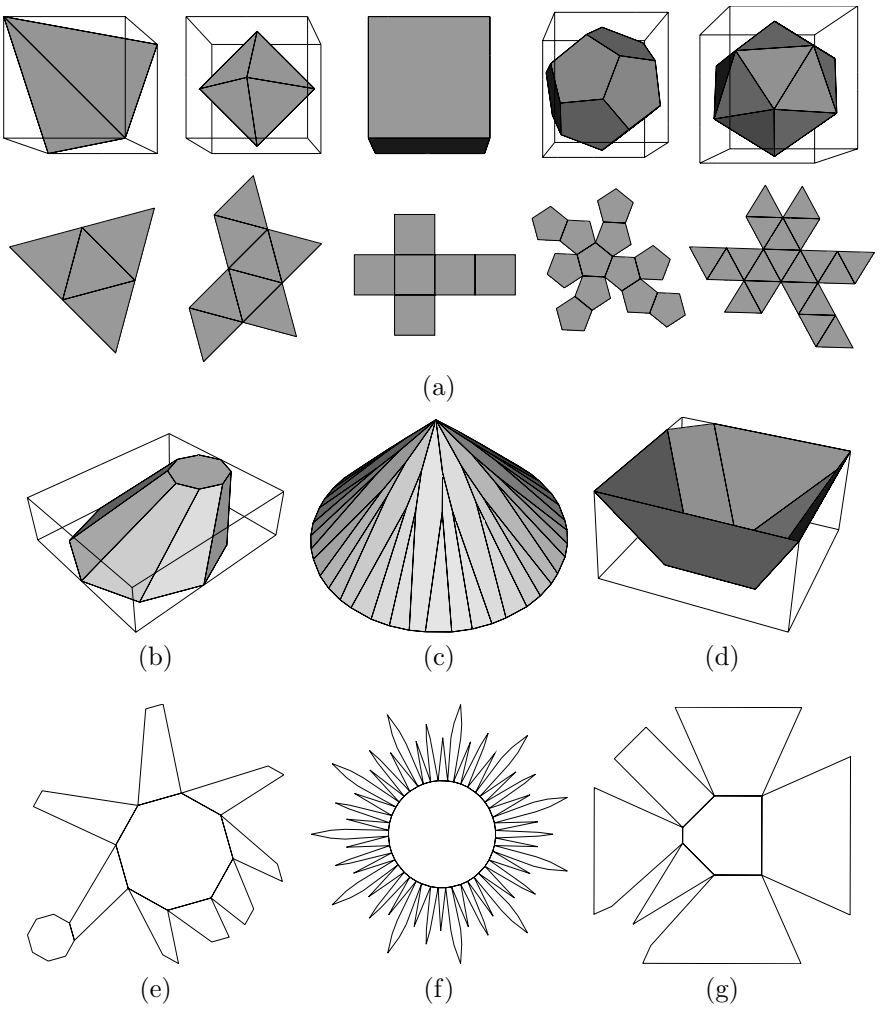


Figure 1.7: Some of the classes of convex polyhedra and polyhedral subsections which are known to be unfoldable:
(a) The platonic solids (Dür25)
(b,e) Prismoids (DO05)
(c,f) Domes (BO07)
(d,g) Volcano unfoldings for 1-ring convex caps (Pin07)

sides are held fixed, then the remaining side must stretch. The lemma, as quoted by Singer (Sin98, p.110), is:

Lemma 1.5 (Cauchy's Arm Lemma) *Suppose we transform a convex (spherical or planar) polygon $A_1A_2 \dots A_n$ into another convex polygon $A'_1A'_2 \dots A'_n$ in such a way that the lengths of the sides A_iA_{i+1} remain unchanged. If the angles at the vertices $A_2, A_3, \dots, A_{n-1}$ remain unchanged or increase, then the length of the remaining side A_nA_1 must also increase.*

The interested reader will find several demonstrations of this lemma in the letters of Schoenberg and Zaremba in (SZ67).

Steinitz Like the Poincaré formula, *Steinitz' Theorem* (Wei99) is one of the fundamental results in mathematics.

Theorem 1.6 (Steinitz) *The one-skeleton of an arbitrary convex polyhedron is a planar 3-connected graph [and each planar 3-connected graph is polyhedral.]*

Several different proofs of Steinitz' theorem have been advanced over the years, but perhaps the most natural demonstration is also the most hands-on: one need only shine a light from near one face of of any transparent convex polyhedron onto a blank wall, to see that the shadows of its edges form a 3-connected graph. David Eppstein's Geometry Junkyard (Eps05) has an excellent ray-tracing of this figure.

Counting spanning trees Read and Tarjan (RT75) provide the following result on spanning trees:

Lemma 1.7 (Read and Tarjan) *A connected graph $G = (V, E)$ has at least 2^t spanning trees, where*

$$t = \left\lceil \frac{-1 + \sqrt{1 + 8 \cdot (\|E\| - \|V\| + 1)}}{2} \right\rceil$$

Counting sides of faces From the Poincaré formula follows a useful result on the number of edges of a face of a polyhedron, formulated by Malcolm Sabin[10]:

Lemma 1.8 (Sabin) *Every closed finite polyhedron of genus 0 must have at least one face of five or less sides.*

Proof. The total number of edges E is equal to half the number of faces times s, the average number of sides, and also half the number of vertices times v, the average valency. Thus $2E/s + 2E/v - E = 2$ and $2/s + 2/v - 1 > 0$. Because a face must have at least 3 sides (and a vertex at least 3 neighbors), we have separate upper bounds on s and v: $2/s > 1 - 2/v \geq 1 - 2/3$, so that $s < 6$. The inequality is strict, the limit value being achieved only for the unbounded hexagonal grid. Similarly $v < 6$. If the average number of sides is less than six there must be at least one face with five or fewer sides. □

[10] Personal communications, 2008

1.5 New Research and Results

The core of this dissertation is separated into four chapters, each of which addresses a different aspect of the unfolding problem.

Chapters 2 and 3 explore the unfoldability of convex polyhedra; the breadth of material is such that it has been divided into two parts. Chapter 2 reviews fundamental conjectures and new negative results, designed to lay the groundwork for a broader understanding of the issues faced in proving that all convex polyhedra are unfoldable–or not. Chapter 2 introduces new approaches and recent research directions which have *not* born fruit, for much can be gleaned from such negative results. In contrast, Chapter 3 describes three distinct possible proofs that all convex polyhedra are unfoldable; each proof is presented, with supporting evidence and (where appropriate) counterexamples; and while none of the three proofs is as yet conclusive, they provide strong directions for future research.

Chapter 4 introduces a series of *Unfolders*, algorithms which try to unfold surfaces both convex and nonconvex. Often expressed in software, the intent of these tools is not to always succeed–there are known examples which prove that they cannot always do so–but rather to make a 'best effort': to unfold surfaces which are unfoldable or to fail gracefully on those which are not and, most importantly, to do so in reasonable time.

Chapter 5 discusses the *coolinoid*, a nonconvex surface with particularly interesting traits. The class of polyhedra known as coolinoids contains surfaces which are developable and also surfaces which are not; detailed analysis of the coolinoid, its shape and related developability, yields curious and intriguing results.

1.5.1 Are all convex polyhedra unfoldable?

There are two sides to the story of convex unfoldability, like players in a game of chess.

On one side of the board, playing White, is the quest for the proof that all convex polyhedra are unfoldable, and certainly there are many points supporting that argument; but none are conclusive, and so across the board sits the opponent playing Black, who seeks an ununfoldable convex surface–the counterexample to White's conjecture.

Arrayed before White are a number of conjectures, though many of the early sallies are easily disproved. White must wrestle with the fact that even though every example it produces is unfoldable, there is as yet no proof that every polyhedron it might *ever* produce is unfoldable. There are a few ways that White can approach the proof:

Classification Defining classes of convex polyhedra which can be shown to be unfoldable, and then eventually showing that all convex polyhedra fit into one of these classes. This approach has been used extensively in previous research, such as prismoids and domes.

General proof Attempt to support the core premise directly. This head-on attack motivates the conjectures of Section 2.2.

Similarity Applying knowledge from the analog case to the discrete. It is well-known that every convex polyhedron can be generally unfolded without overlap; Section 2.3 seeks to leverage this knowledge for edge-unfoldings.

Cut-graph construction Showing that if a cut-graph has certain traits, it will unfold the polyhedron without overlap; and then showing that such a cut-graph can be constructed for every input. Convex curves, local convexity and the alpha-beta rules (Sections 2.5, 2.6 and 3.3) each take different approaches to this goal.

Reduction Beginning from a polyhedron which is known to be unfoldable, describe a method to transform the source polyhedron into one whose unfoldability had not been known previously; if unfoldability is an invariant under the transformation then the descendant's developability may be guaranteed. Then, show that the transformation method applies to every convex polyhedron. This philosophy motivates the Leaf-Node Truncation theorem of Section 2.4.

Net construction Showing that there exists some algorithm which can always build an intersection-free net from the faces of any convex polyhedron, and that this algorithm must always apply to every input; one such candidate is the Anchorable Convex Hull (Section 2.7).

Error correction An extension to either form of construction is a hybrid of the two: to build an unfolding with 'controlled' amount of overlap and to then repair its results in a second pass. This method is discussed in Section 3.4.

The constructive approaches–net or cut-graph construction–are excellent candidates for expression in software. An algorithm whose complexity makes correctness difficult to prove can still be strongly supported by experimental evidence; this very different problem is addressed in Chapter 4.

Meanwhile, Black prepares a very different sort of argument. Black's goal is to show that not all convex polyhedra are developable. The easiest way to do so, of course, would be to present an actual counterexample: some closed, convex, n-sided surface without a single valid unfolding. Thus far, there are few clear paths to such a construction. Section 2.8 describes one step in that direction: *polyhedral banding* is a method for constructing a convex polyhedron which is 'less' unfoldable than others, that is, is demonstrably likely to have fewer total possible unfoldings than a randomly-generated convex polyhedron of similar degree. If the ununfoldable counterexample *does* exist, exploring methods such as polyhedral banding could provide essential insights into its discovery.

1.5.2 Unfolding algorithms

Beyond mathematical proof for convexity lies another, altogether different sort of problem: algorithmic solutions for the general case. The two questions are not unconnected: the perfect unfolding algorithm, which provably could not fail, would constitute a proof by existence of convex developability.

Chapter 4 explores a suite of unfolding techniques, discussing methods ranging from brute-force and randomized search, through simple heuristic algorithms, to sophisticated curvature-driven parallelized models.

It is worth noting that Black has a vested personal interest in these algorithms. If Black wishes to claim that some particular polyhedron cannot be developed without overlap, the burden of proof will be much lighter if an algorithm has been found which can short-cut the enumeration without loss of information.

1.5.3 Unfolding the coolinoid

In the course of exploring algorithms for unfolding non-convex surfaces, the author encountered a very interesting special case: the *coolinoids*. A coolinoid is a surface of uniformly negative internal angle deficit with two borders. What is interesting about the coolinoid is, first of all, that it is unfoldable at all; secondly, that it has the least number of border loops of any surface of uniformly negative internal angle deficit; and thirdly, that some coolinoids are developable but others are not, and the border between the two classes follows a subtle and fascinating form.

The author's study of coolinoids formed the basis for the paper *A Developable Surface of Uniformly Negative Internal Angle Deficit* (Ben07), presented by the author at Mathematics of Surfaces XII, 2007. The coolinoid is discussed in detail in Chapter 5.

1.6 Experimental Apparatus

1.6.1 Software

The YAMM software package has been designed as a testbed for unfolding concepts. Running on Windows operating systems and written in C++, it models high-polycount surfaces in 3D with OpenGL; it provides a context in which the user is free to experiment with various implementations of unfolding algorithms, exploring the effects of different models and methods.

YAMM provides a host of research-oriented features, such as data validation, dataset repair, real time polygon mesh editing, statistics tracking, automated Monte-Carlo search, and output to PDF and OFF. The YAMM software is described in detail in Appendix B and available online.

YAMM has proved absolutely invaluable in the visualization of problems, in the evaluation of conjectured approaches, and in the discovery of counterexamples. A number of the models presented here were designed and assessed entirely within YAMM.

1.6.2 Datasets

A large number of polyhedral models were collected or constructed in the course of the work presented here, covering special cases and interesting surfaces both convex and non-convex. They are listed in Appendix C and available online.

Chapter 2

Groundwork in Unfolding Convex Polyhedra

2.1 Introduction

This chapter explores the author's early work on the question of whether or not all convex polyhedra are unfoldable. The concepts and insights found here form essential underpinnings to the material presented in Chapter 3.

In a sense, every result in this chapter is negative; but to dismiss such results as negligible would be a great mistake. The tree of possible avenues of research is broad and deep, and each negative result presented here acts to trim that tree. Every category of cut-graph which is shown *not* to unfold all convex polyhedra, every algorithm which fails to generate an overlap-free net, marks another branch which may be pruned from future research. Furthermore, each failed avenue often offers up new insights; the Anchorable Convex Hull, for example, inspires several of the proofs proposed in the next chapter.

This chapter is organized as follows:

- Section 2.2 explores, and rapidly discards, early and naïve conjectures about the problem. By the end of section 2.2 it should be clear that the problem is not going to be quickly answered. (By the end of the chapter, it will be clear that finding an answer may be quite difficult.)
- Section 2.3 answers Fukuda's conjecture as to whether an unfolding along shortest-path edge cuts will always be free of overlap.
- Section 2.4 describes the author's work with Joseph O'Rourke for the paper *Unfolding Polyhedra via Cut-Tree Truncation* (BO07), presented to the Canadian Conference on Computational Geometry 2007. This section proves the unfoldability of the *domes*, a class of convex polyhedra.
- Section 2.5 provides a counterexample to the conjecture that any cut-graph which is an open convex curve must develop without overlap.
- Section 2.6 presents the author's proof for the *locally-convex cut-graph*. It is shown that every convex polyhedron whose 1-skeleton can be spanned by a locally-convex cut-graph is unfoldable without overlap.

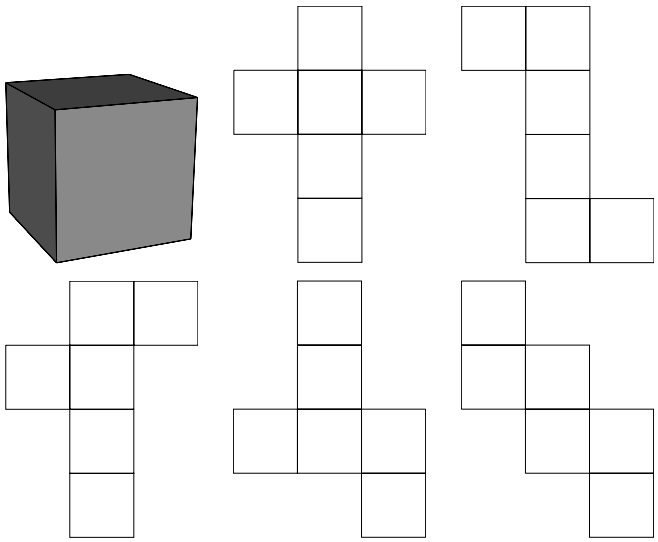


Figure 2.1: A cube and several of its unfoldings

- Section 2.7 introduces the *Anchorable Convex Hull* ('ACH'), an iterative algorithmic proof which attempts to show that a conflict-free partial unfolding may be extended to a larger partial unfolding without sacrificing the guarantee that there is no overlap. In doing so the ACH introduces the idea that a partial unfolding may be held convex, growing outwards from some central point; this theme is revisited in Chapter 3.
- Section 2.8 explores the construction of *banded polyhedra*, a demonstrably 'hard to unfold' class of polyhedra. Some material in section 2.8 is drawn from the paper *A Class of Convex Polyhedra with Few Edge Unfoldings* (BO08), co-written with Joseph O'Rourke.

2.2 Early Conjectures

2.2.1 The most primitive conjecture: all convex polyhedra

A cube is completely unfoldable: no unfolding of a cube has overlap. From this one might generalize the conjecture that no unfolding of a convex polyhedron will ever overlap. (Figure 2.1.)

Conjecture 2.1 *Every unfolding of a convex polyhedron will be without overlap.*

Rebuttal
Many counterexamples have been found of convex polyhedra which possess at least one overlapping unfolding. In fact, it has been found that as the complexity (number of faces) of a polyhedron rises, that the odds of a randomly-chosen cut-graph unfolding the polyhedron rapidly approaches zero; see Section 4.9 and (O'R00, p.3).

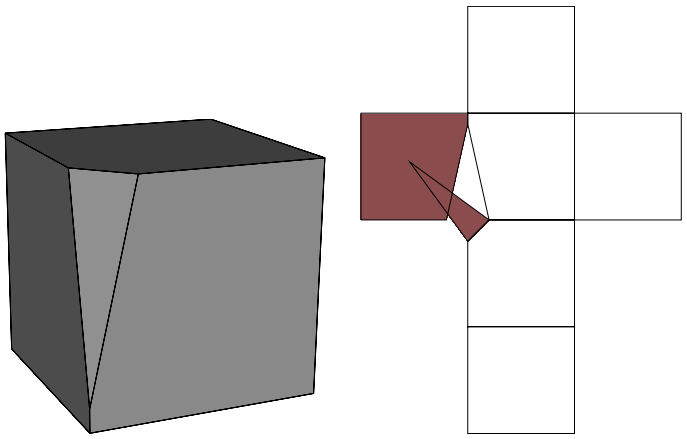


Figure 2.2: A degree-3 truncation of the cube creates a seven-sided figure in which 100 of the 1,682 possible cut-graphs will unfold to overlap

A common counterexample to the argument of the cube is the truncated cube (Figure 2.2.) While many of the unfoldings of the truncated cube retain the cube's property of non-overlap, there are 100 unfoldings (out of a total of 1682) which self-intersect. All of these illegal unfoldings share the common attribute that the cut-graph cuts through the two long edges of the truncation face and leaves the short edge intact.

The example of the truncated cube is a commonly cited model from the `Mathworld` online resource. Other, similarly simple and intuitive examples include Namiki and Fukuda's slim tetrahedron (Figure 1.5, p. 13) and the truncated tetrahedron, a minimal model found in the course of preparing (BO07) which demonstrates the same behavior as the truncated cube and does so with fewer faces. The truncated tetrahedron shown in Figure 2.3 can unfold to self-intersection, but only in three out of 75 cases. In each of the three self-intersecting unfoldings, the tetrahedron is cut by a cut-graph which contains both of the long edges of the truncation face. □

2.2.2 Simplicial convex polyhedra

Given that not all convex polyhedra are always unfoldable and that this has been demonstrated by counterexamples with non-simplicial faces, one might suppose that the lesser class of *simplicial* convex polyhedra are always unfoldable.

Conjecture 2.2 *Every unfolding of a simplicial convex polyhedron will be without overlap.*

Rebuttal
The slim tetrahedron devised by Namiki and Fukuda (NF94) (Figure 1.5) is a simplicial (and minimal) counterexample. The model has sixteen distinct unfoldings, two of which self-intersect.

□

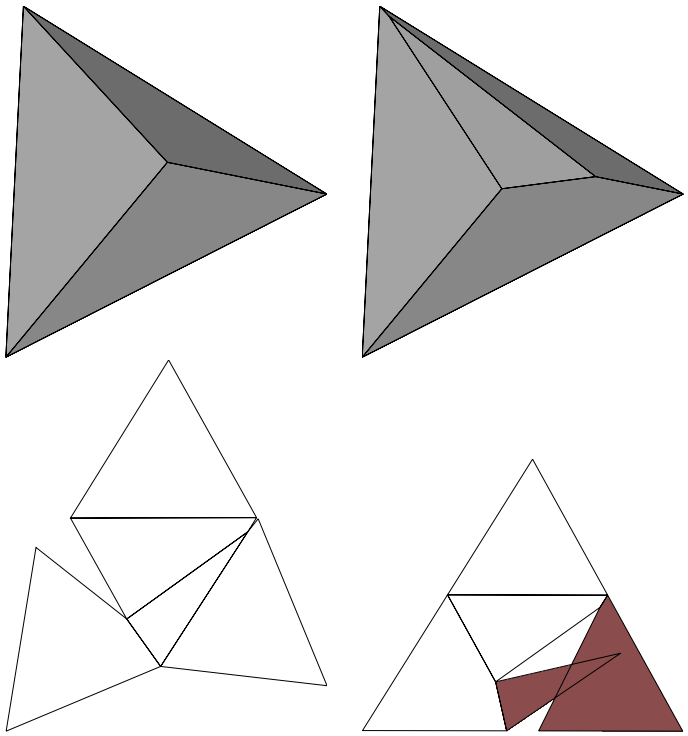


Figure 2.3: A degree-3 truncation of the tetrahedron creates a five-sided figure in which three of the 75 cut-graphs will unfold to overlap

2.2.3 'Sufficiently smooth' convex polyhedra

The slim tetrahedron counterexample is distinguished by two vertices of extremely high angle deficit. It would not be unreasonable to conjecture that simplicial convex models without such sharp vertices might be always unfoldable. Figure 2.4 shows a sphere S of such high face count that at every vertex S is within some ϵ of being flat. (Note that by increasing the face count, ϵ may be made arbitrarily small.) S is shown unfolded without intersection in Figure 2.4.

Conjecture 2.3 *Every unfolding of a simplicial convex polyhedron of sufficiently low maximum angle deficit will be without overlap.*

Rebuttal

It is clear in examining the unfolding of S shown in Figure 2.4 that even a simple deviation from the unfolding shown could result in conflict. Selecting an unfolding at random, a counterexample is found almost trivially; in fact, every random unfolding of the high-polycount sphere generated in the course of this research was self-intersecting (Figure 2.5.)

A self-intersecting unfolding may also be constructed deliberately. To induce conflict, a cut-graph is chosen in which a cut edge 'curls' on the surface of the sphere, deviating sufficiently from a locally-geodesic curve. Figure 2.6 shows a high-polycount sphere in which cuts in the cut-graph used in Figure 2.4 have

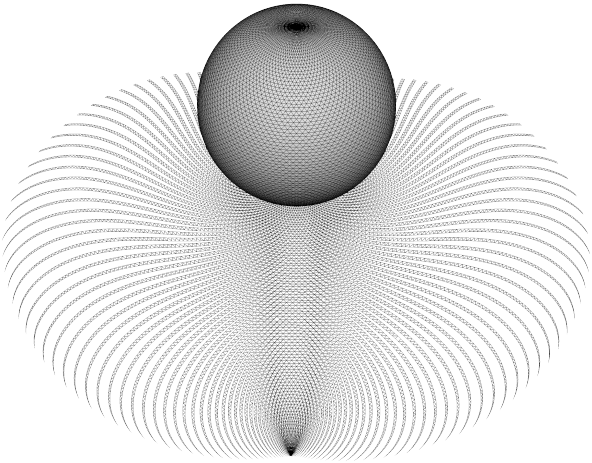
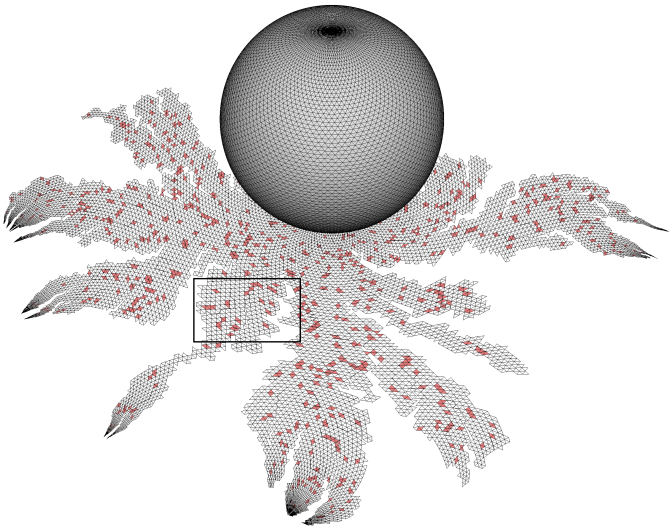


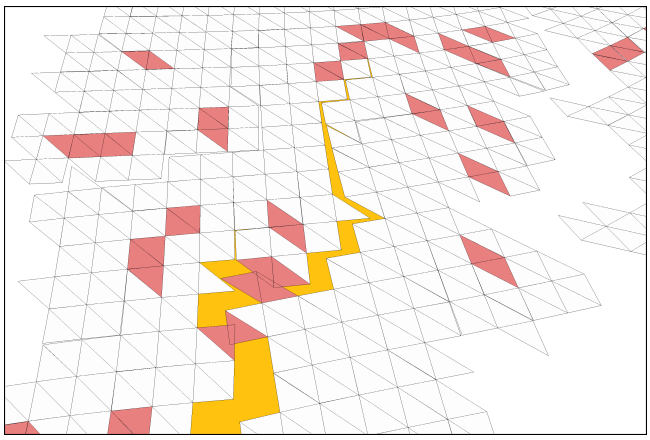
Figure 2.4: A sphere with low local curvature unfolded without intersection

been interrupted and bent to one side, forced into an L-shaped hook. This causes an immediate conflict with adjacent geometry.

□

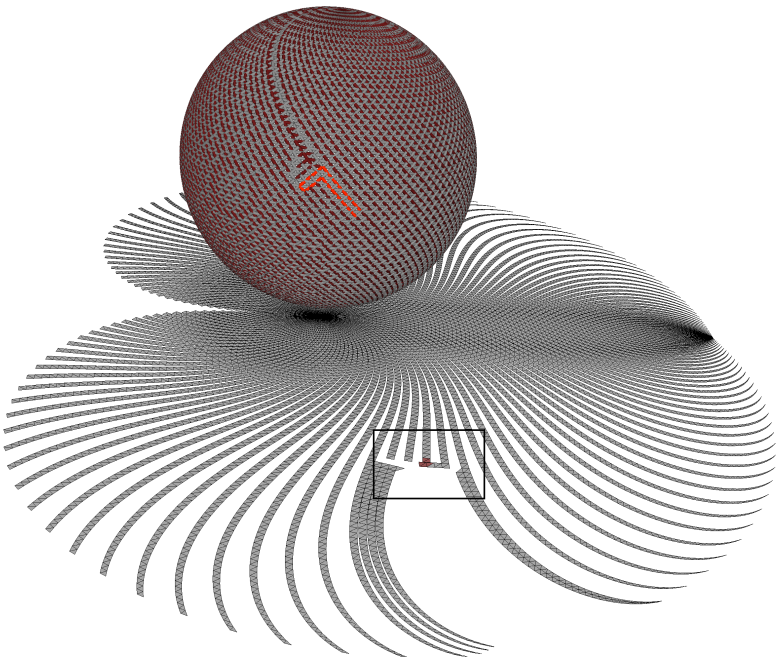


(a)

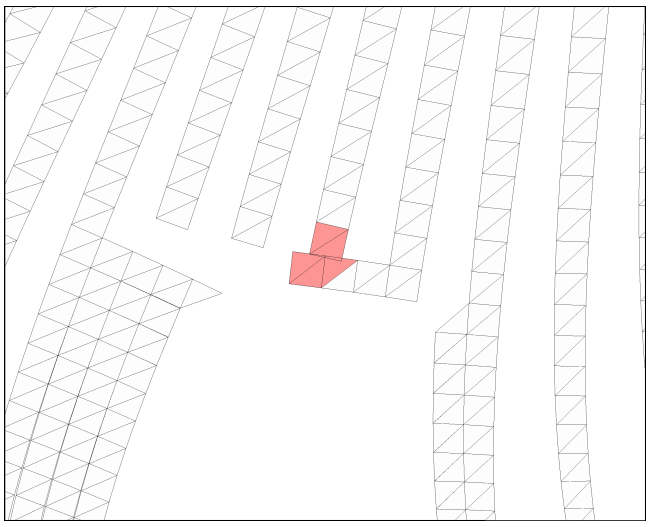


(b)

Figure 2.5: (a) A sphere with low local curvature unfolded with random cuts, demonstrating conflict independent of local curvature; small angle deficits lead to conflicts several cut edges away. (b) Detail showing the widening 'chasm' between two sides of a cut. As the two sides separate, faces intersect and clash.



(a)



(b)

Figure 2.6: (a) The low-local-curvature sphere, cut to deliberately self-intersect. (b) Detail of the self-intersection in the unfolding plane.

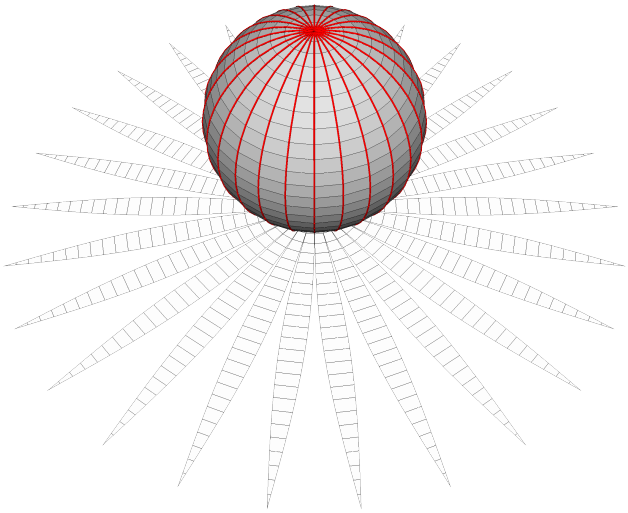


Figure 2.7: A sphere edge-unfolded along polyhedral geodesics

2.3 The Discrete Star Unfolding

It is shown in (AAOS90) that for any polyhedron P, given any point x on P chosen such that there is only one shortest path from x to each vertex of P, the *star unfolding* found by cutting P along the geodesics from x to each vertex will be without overlap.

In light of this, it is reasonable to conjecture that any edge-unfolding which is 'close enough' to a star unfolding will be without overlap. This conjecture was first postulated by Komei Fukuda in (Fuk97), who asked,

> Does a shortest-path spanning tree of a convex polytope induce a non-selfoverlapping unfolding?

This conjecture is encouraged by the insight that many successful unfoldings, especially as surfaces become more complex, seem to favor cut-paths which travel along relatively straight lines. For example, consider Figure 2.7, in which a sphere is unfolded successfully by cutting along geodesic paths from the topmost vertex of the sphere to every vertex below with a single edge cut to the bottom-most vertex.

Efficient construction of the star edge-unfolding

Dijkstra's Algorithm (Dij59) is an $O(n^3)$ method for finding the shortest path on a graph. However, Dijkstra cannot be applied directly to the task of finding shortest paths on a polyhedron because the crossing points are not known *a priori*. Dijkstra's algorithm is, by nature, constrained to operate on a graph whose edges have fixed known weights.

A *Steiner point* is an extra vertex added to a graph which is not a part of the initial input.

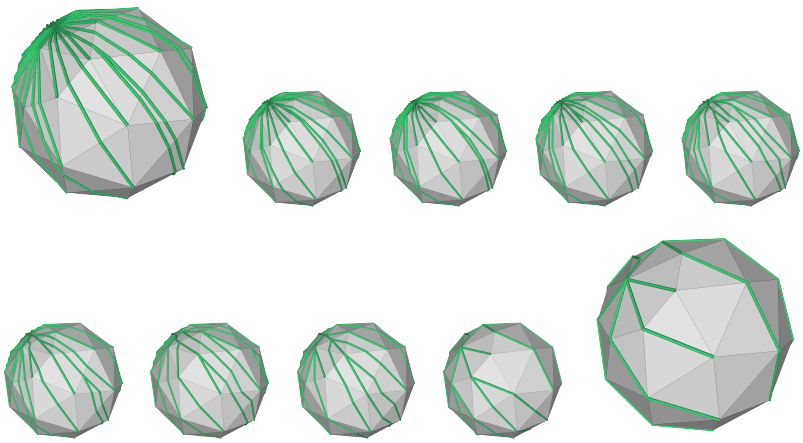


Figure 2.8: Progressive reduction of the maximum number of Steiner points per edge in the approximation of the shortest path on a polyhedron P. The leftmost sphere shows shortest paths approximated with up to 20 Steiner points per edge; the rightmost sphere has none.

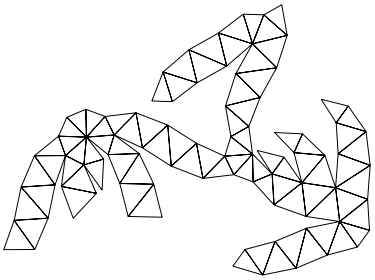


Figure 2.9: The unfolding of the final stage of P from Figure 2.8.

In (LMS97; ALMS98), Lanthier et al. propose a mechanism for the rapid approximation of the shortest path on a polyhedron by adding a set of m Steiner points along each edge. Through some dexterous juggling of constants and the use of an ϵ-error constant, they are able to bring the time to find a shortest path on a polyhedron down to $O(mn \log mn)$.

The Steiner points method has inspired the approach presented here for the construction of an edge unfolding which most closely follows a star unfolding. A star unfolding is initially calculated by using a high number of Steiner points. This 'ideal' unfolding is then re-created without Steiner points, following only edges along the shortest paths between vertices. The resulting coarse graph approximates the star unfolding without cutting faces. Figure 2.8 shows a gradual reduction (displayed in stages for clarity) from fine to coarse graphs, ending in a cut-graph which then unfolds to the figure shown in Figure 2.9.

Fukuda's conjecture

It has now been demonstrated that an edge-unfolding derived from a star unfolding can be found which is without overlap. It would then seem reasonable to ask whether this must always be the case, for all such edge-unfoldings.

Conjecture 2.4 (Fukuda) *Any edge-unfolding which follows closely enough to a general star unfolding will be without conflict.*

Rebuttal

Critical to the rebuttal to this conjecture is the notion of 'follows closely enough'. The shortest edge-path from one vertex to another may actually be a small distance from the true shortest path over the surface between the two vertices. The star unfolding also assumes an unspecified 'source' vertex, which (when seeking a rebuttal) may be chosen with an eye to inducing pathological behavior.

In his 1997 thesis *Nets of Polyhedra* (Sch97), Wolfram Schlickenrieder describes the counterexample to Fukuda's conjecture, found by evaluating a cut-graph of shortest-path edges on a randomly-generated polyhedron. The author of this dissertation arrived independently at a similar conclusion, with the following result:

A convex polyhedron P was generated by taking the intersection of 77 infinite half-planes. Figure 2.10(a) shows a star unfolding of P while 2.10(b) shows the graph of edges on P which most closely fits the star unfolding. Figure 2.10(c) shows the unfolding of P according to the chosen cut-graph: there are two self-intersections. Figure 2.10(d) shows one such intersection in detail.

This demonstrates that not every edge-unfolding which most closely follows a general star unfolding will be without overlap. □

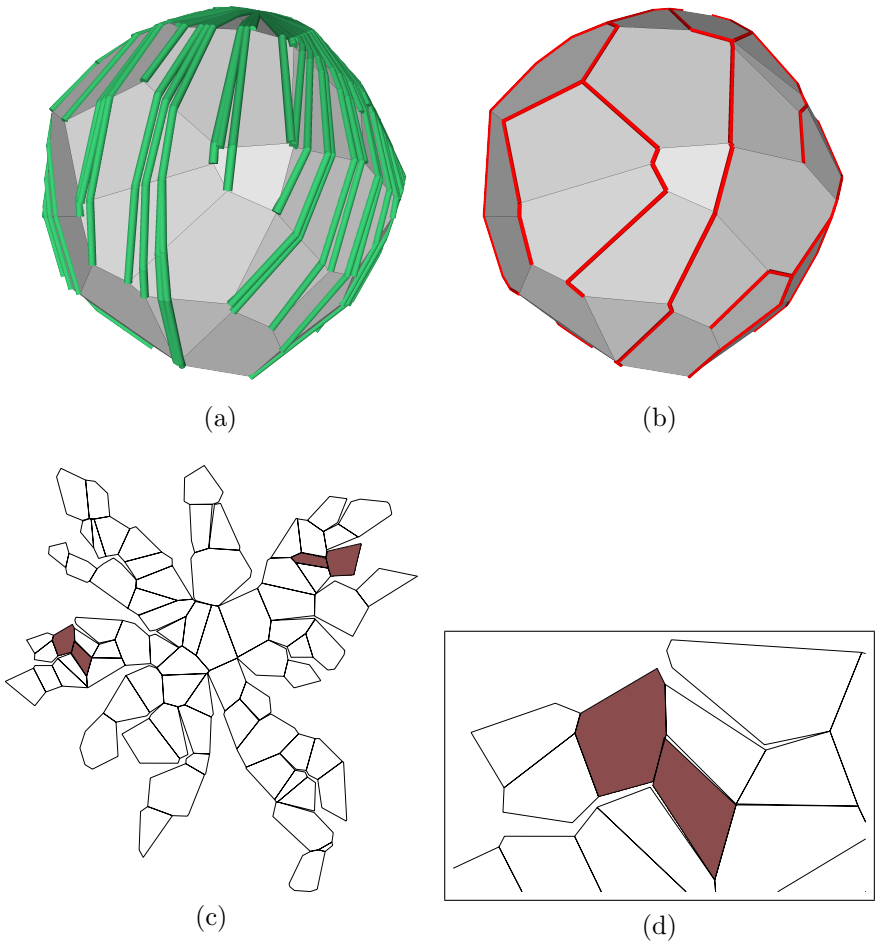


Figure 2.10: A randomly-generated 77-faced polyhedron showing conflict when unfolded with a cut-graph which most closely fits a star unfolding. (a) General unfolding cuts. (b) Best-fit edge paths. (c) Unfolded net, with overlap. (d) Detail of unfolding overlap.

2.4 Cut-Tree Truncation

The author presented the paper *Unfolding Polyhedra via Cut-Tree Truncation*[1] (BO07), co-written with Joseph O'Rourke, to the Canadian Conference on Computational Geometry 2007. Cut-tree truncation is a method by which one polyhedron may be transformed into another while retaining prior knowledge of the unfoldability of the polyhedron. For example, if some polyhedron P_0 is known to be unfoldable by some cut-graph T_0, cut-tree truncation shows how a new polyhedron P_1 and cut-graph T_1 may be derived which are also known to unfold without overlap. This section discusses the key ideas and theorems of the paper.

2.4.1 Definitions

Two key terms are defined: the *empty sector property* and the *degree*-3 *leaf truncation* operation.

Empty Sector When a net is unfolded by a cut-graph, the developments of the edges incident to each leaf of the cut-graph unfold to open into an arc in the plane (Figure 2.11(a) and (b).) An unfolding has the *empty sector property* if no other polygon intrudes into the arc of any leaf-node vertex.

This definition can be generalized to the surface by saying that a surface P has the empty sector property if there exists any cut-graph T such that (P, T) has the empty sector property.

Leaf Truncation Let x be a leaf of the cut-graph T, with y the parent of x in T. A *leaf truncation* is the removal by truncation of a vertex x from P where x is a leaf node of T, yielding P' with a new face and T' in which x has been replaced by a tree of depth one of new edges rooted in x's former parent (Figure 2.11(c) and (d).)

A *degree-3 leaf truncation* is the truncation of a trivalent leaf-node vertex x, yielding P' with a new triangular face $\triangle abc$, $a \in xy$. In T' x has been replaced with the Y-fork of edges ya, ab and ac.

A degree-3 truncation is the only vertex truncation which can produce a tree of edges rooted at y; the new cut edges must trace the border of the new face, dictating that any higher-degree truncation cannot replace x with a tree of depth one without splitting the newly-added face into several triangles fanning from a. Therefore Theorem 2.5 is restricted to degree-3 truncations.

[1] The following passage appears in the Collaboration Statement which accompanies this dissertation:

> *The lion's share of the text was written by Joe, and I produced all figures in the paper except figures 1, 2, and 8; the models used in figures 3, 4, 9 and 10 are of my own design. Joe credits me with the insight that led to section 4, 'Cutting to achieve degree-k vertices'. Section 5, 'Empty sector essential', relies on my convex cap counterexample. Section 6, 'Polyhedra achievable by degree-3 leaf truncation', was the joint result of extensive discussion by email.*
>
> *I presented this paper at CCCG2007 and produced all figures and text of the presentation. As part of the presentation I presented a rewritten version of the proof of the main theorem and of the proof that all domes are unfoldable, both updated for clarity.*

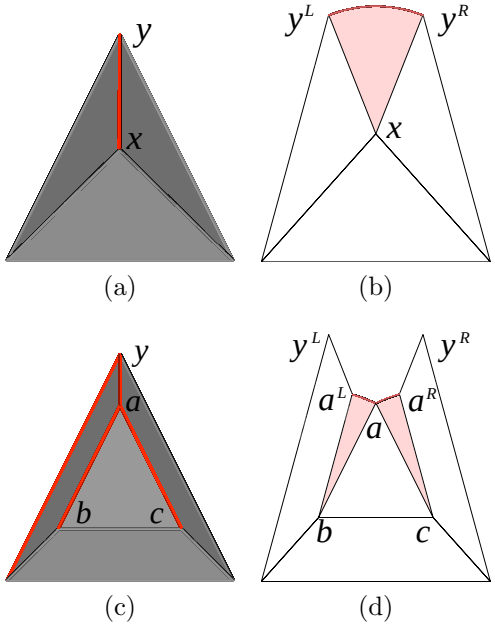


Figure 2.11: A cut edge (a) has the empty sector property at vertex x (b). When x is truncated (c), the cut edges of the new triangle $\triangle abc$ also retain the empty sector property (d).

2.4.2 The cut-tree truncation theorem

The linchpin of (BO07) is the proof that degree-3 truncation preserves unfoldability. The two key results are the following theorem and its corollary:

Theorem 2.5 (Theorem 1, (BO07)) *If a non-overlapping unfolding of a polyhedron P via a cut-tree T has the empty sector property, then the cut-tree T' produced by a degree-3 leaf truncation (a) unfolds P' without overlap, and (b) has the empty sector property.*

The proof of the theorem is an exercise in geometric reasoning. As the construction of the proof was almost entirely the work of O'Rourke, it will not be reproduced in this dissertation; in brief, the proof shows that the development of a, the tip of triangle $\triangle abc$, must fall in the plane somewhere within the arc of rotation between the developments the edge xy. So long as the original arc between xy^R and xy^L was empty, it can be known that the matching arc with endpoints at y^R and y^L in the new unfolded net is also empty and therefore the new triangle $\triangle abc$ develops without overlap.

Corollary 2.6 (Corollary 2, (BO07)) *Any polyhedron P derived by repeated degree-3 leaf truncations from an initial polyhedron P_0 and cut-tree T_0 with the empty sector property, unfolds without overlap via the derived cut-tree T.*

Figure 2.12 demonstrates Corollary 2.6 in action. By Theorem 2.5 any initial unfolding (P_0,T_0) having the empty sector property implicitly defines a broad

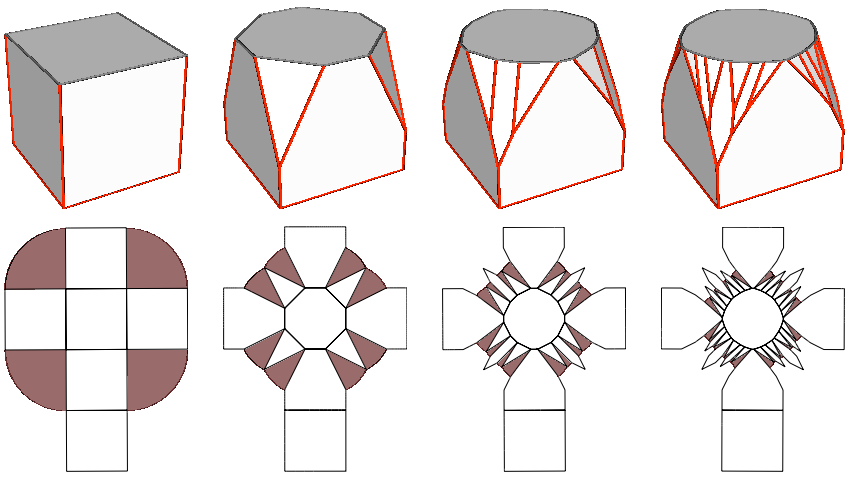


Figure 2.12: The progressive leaf-truncation of a cube to a new developable convex surface

category of more refined polyhedra which also have the property (and are thus unfoldable.)

However, it has proved difficult to extend this broad category to cover every convex polyhedron (which was the original goal.) Leaf truncation can only be applied to leaves of polyhedra and the theorem requires that every such leaf be of degree 3; this partially restricts the theorem to polyhedra with trivalent vertices and partially restricts it as well to those with triangular faces (i.e., faces which can be achieved through the truncation of a trivalent vertex.) This is not to oversimplify by saying that the theorem applies only to trivalent simplicial polyhedra; vertices and faces of degrees other than 3 are still acceptable, so long as they are not the targets of a truncation operation.

2.4.3 Applications of the theorem

From Corollary 2.6, several further insights are derived:

In Section Three of (BO07), a counterexample is given of of a broad-based five-sided dome whose apex cannot be leaf-truncated (Figure 2.13.) After the truncation of the tip of the five-sided pyramid P^t, the newly-created pentagonal face f cannot be glued to any of the new edges without overlap. This demonstrates conclusively that Theorem 2.5 does not extend beyond the truncation of trivalent vertices.

Conversely, in Section Four it is demonstrated that an ϵ-close truncation can generate a polyhedron with vertices of degree > 3 which has the empty sector property. Thus if one seeks to prove the developability of non-trivalent polyhedra, it is at least partially possible to achieve them through truncation. However, Section Four cautions the reader that the use of an ϵ does create a new edge in the face-to-face connectivity graph of P which had not previously existed and which does not vanish as $\epsilon \to 0$, but it "should". Care must thus be taken that this new length-ϵ edge never be left uncut; if it were, the two faces at either side would be attached only by geometry which rapidly converges to a

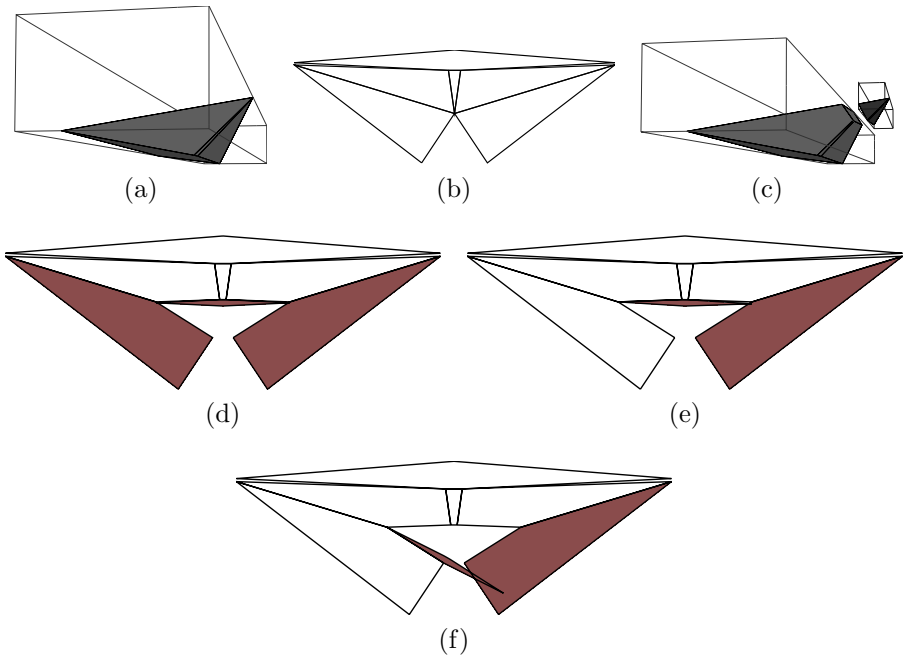


Figure 2.13: (a) Pyramid P with five-sided base. (b) The pyramid, unfolded. (c) The truncated pyramid P^t with tip removed. (d), (e), (f) The five possible attachments of the new pentagonal face f to P^t lead to overlap (two symmetric unfoldings not shown.)

single point, violating the rules of the edge-unfolding.

Section Five gives an example which demonstrates that the empty sector property is essential. A convex cap is constructed (Figure 2.14) which, when cut by a specific cut-graph, does not have the empty sector property but is still unfolded without overlap. A degree-3 leaf-truncation is then applied to one vertex of the cap and a Y-fork inserted into the cut-graph; the resulting figure is no longer free of overlap. This figure was originally conceived by the author as a counterexample to the proof of the *smallest ununfoldable polyhedron*, which is discussed in Section 3.2.

Condensed in the printed version of the paper was another counterexample, of a truncated tetrahedron which demonstrated that the Y-fork defined in the degree-3 leaf truncation was also essential to the theorem (Figure 2.15.)

2.4.4 Domes

The proof that all domes are unfoldable is then presented:

Theorem 2.7 (Theorem 3, (BO07)) *Starting from a pyramid P_0 and cut-tree T_0 the star of edges incident to the apex a of P_0, the polyhedra achievable via degree-3 leaf truncation are all domes. And conversely, every dome (except possibly a wedge) can be realized by a series of degree-3 leaf truncations from some pyramid P_0.*

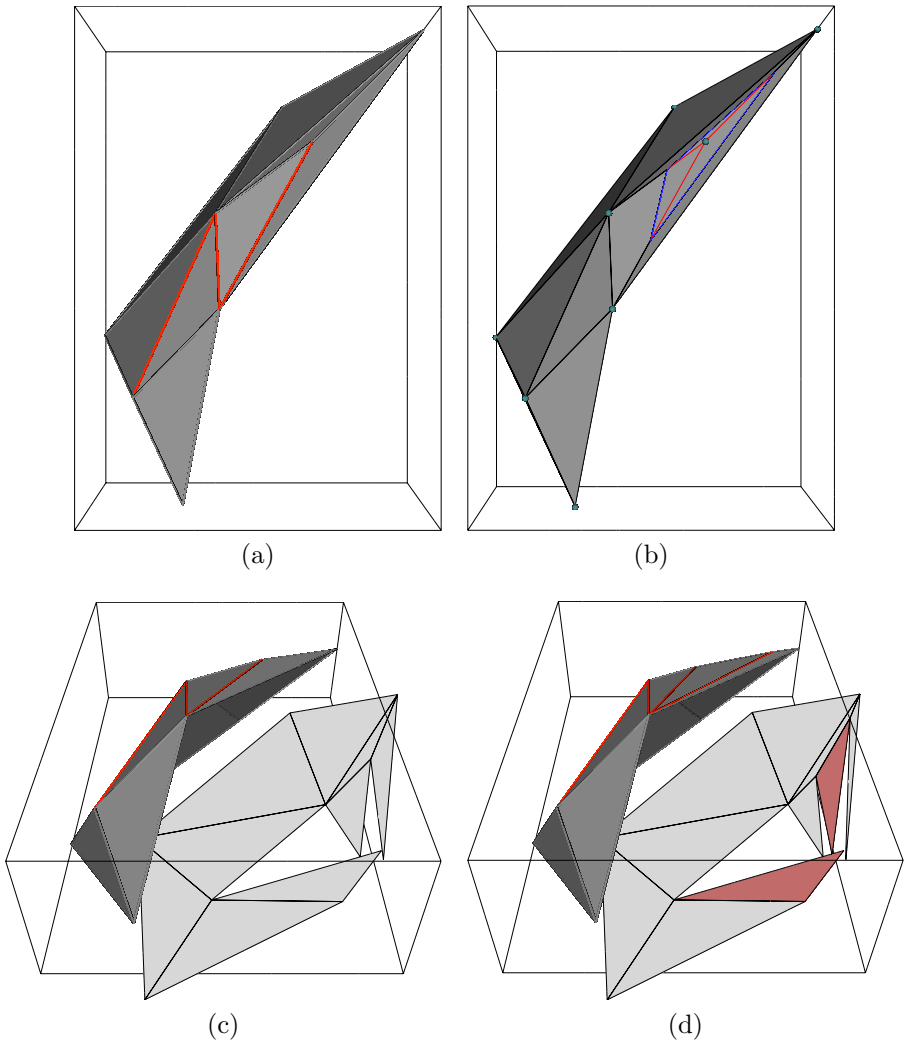


Figure 2.14: Without the empty sector property, degree-3 vertex truncation cannot guarantee non-overlap. (a) The convex cap counterexample, shown from above, with cut-graph in red (b) A single vertex in truncated and a Y-fork introduced (c) The untruncated cap shown in perspective, positioned above its unfolding (d) The cap after truncation; the unfolded net now self-intersects.

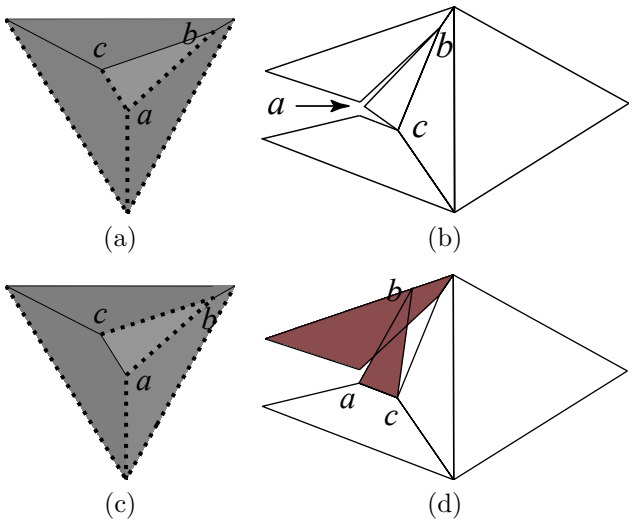


Figure 2.15: Including the edge bc and thus failing to create a Y-fork in the degree-3 leaf truncation can lead to overlap.

Proof. The proof is given in two parts: from pyramid to dome and from dome to pyramid. Wedges are excluded as they do not fit the mechanism of the proof, but the unfoldability of a wedge is straightforward.

Pyramid → Dome Given a pyramid P with base face B, let i index the i-th truncation of P. The leaf nodes of T_i all lie on B_i. Every degree-3 truncation will intersect B_i and create two new vertices which also lie on B_i. Thus every face created will have at least two vertices on B ensuring that P_i is a dome.

Dome → Pyramid Let D be a dome with base face B. There must be at least one triple of sequential edges a, b, c on B such that the extensions of a and c cross outside x of B. Extending B and the two faces at a and c until they meet at x, a new dome D_1 with one fewer vertex on its base face B_1 is created. Continuing in this manner will generate a dome D_k with a triangular base B_k. Clearly D_k must be a tetrahedron.

□

The proof given of Theorem 2.7 is an arguably simpler proof of the unfoldability of the domes than that given in (DO07).

2.4.5 Extensions to leaf-node truncation

(BO07) defines the empty sector property in terms of the entire surface, but it is worth noting that the property is purely local in nature. It would be just as accurate, and more powerful, to say that individual leaves of the cut-graph can be said to have the empty sector property.

This has interesting implications for the description of the *smallest undevelopable polyhedron* (section 3.2.)

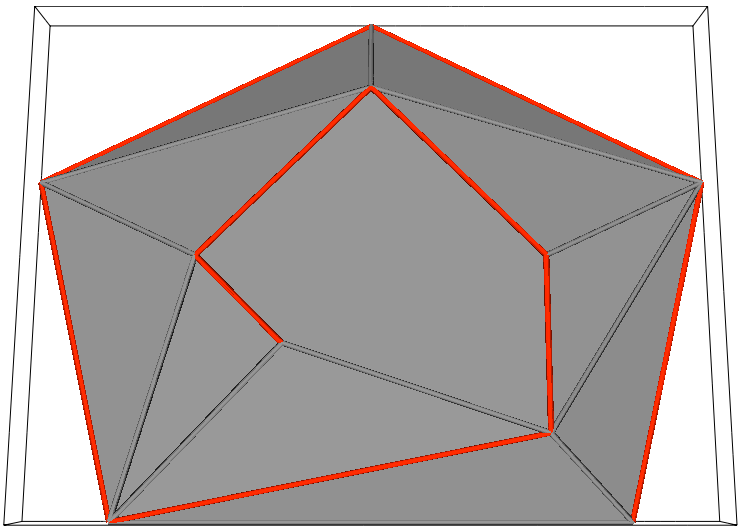


Figure 2.16: A spiral cut ($\lambda(v) \leq \pi \forall v \in P$, shown in red) on a convex polyhedron with pentagonal base and a rotated, inset pentagonal top

2.5 A Cut-Graph of Convex Curves

Recall from p. 4 that $\lambda(c_i)$ for a vertex c_i on a path is the *left interior angle* of c_i, and that a *closed convex curve* is defined as a path $C = (c_0, c_1, \ldots, c_{n-1})$ on the surface where $\lambda(c_i) \leq \pi$ for all i and $c_0 = c_{n-1}$.

O'Rourke and Schevon show in (OS89) that a closed convex curve on a surface develops without self-intersection, that is, the developments of the left and right edges of the curve will not cross in the plane. This proof leads naturally to the supposition that a cut-graph composed solely of convex curves will unfold the polyhedron without overlap. However, adding the requirement that the curves be closed could, perhaps, be unnecessarily restrictive. Can the closure requirement be removed while still retaining a valid unfolding?

Conjecture 2.8 *A cut-graph in which $\lambda(v) \leq \pi$ for all v cannot self-intersect.*

Rebuttal
Beginning with a carefully-chosen polyhedron P (Figure 2.16,) a simple Hamiltonian cut-graph (without branches) is constructed which turns to the left ($\lambda(v) \leq \pi$) at every vertex. The path forms a spiral on one side of P, leaving the other side (which consists of a single face) untouched. This cut-graph unfolds P to self-intersection (Figure 2.17.)

□

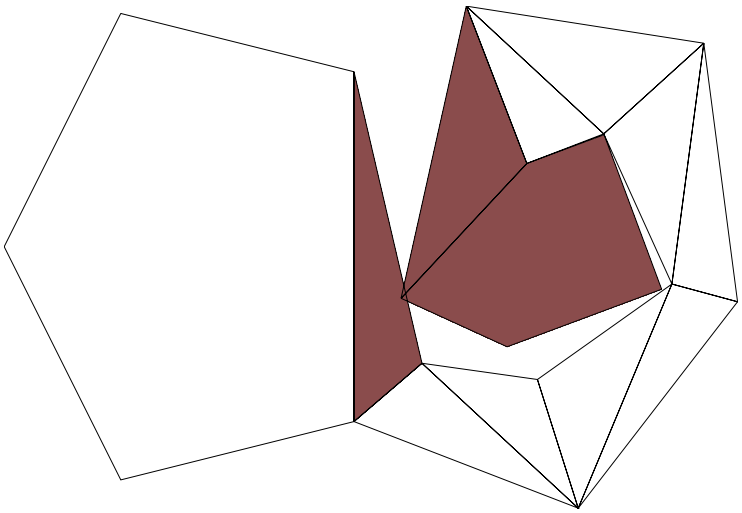


Figure 2.17: Unfolding along the spiral cut leads to overlap

2.6 Local Convexity

This section presents the proof that every convex polyhedron whose 1-skeleton can be spanned by a *locally-convex cut-graph* is unfoldable without overlap. Counterexamples are then given which demonstrate the limits of application of this technique.

2.6.1 Terminology of local convexity

Local convexity is defined as follows:

- If vertex v is a leaf node of cut-graph T, connected to T through neighboring vertex w, where the first leaf node of T encountered after v in a clockwise traversal of the edges of the unfolded net is labeled u and the previous leaf node encountered before v in the counterclockwise traversal is labeled x, then v is *locally convex* if $\angle(vw^L, vx) \leq \pi$, measured counterclockwise in the unfolded plane; and $\angle(vw^R, vu) \leq \pi$, measured clockwise in the unfolded plane.

- If v is not a leaf node of T then v is *locally convex* if none of the totals of the incident angles of the faces between the cut edges of v exceed π. Where v is cut by exactly two edges, this is equivalent to $\rho(v) \leq \pi$ and $\lambda(v) \leq \pi$, i.e., the vertex lies on a convex curve in both clockwise and counter-clockwise directions. (Recall from p. 4 that $\lambda(v)$ and $\rho(v)$ are the *interior angles* of v.)

- A cut-graph T is *locally convex* if every vertex in T is locally convex.

See Figure 2.18. Informally, the definition for a leaf node v states that if the surface were to be cut by a single new edge from v to x, the leaf node after v in a counterclockwise traversal of the edges of the unfolded net, and if that new

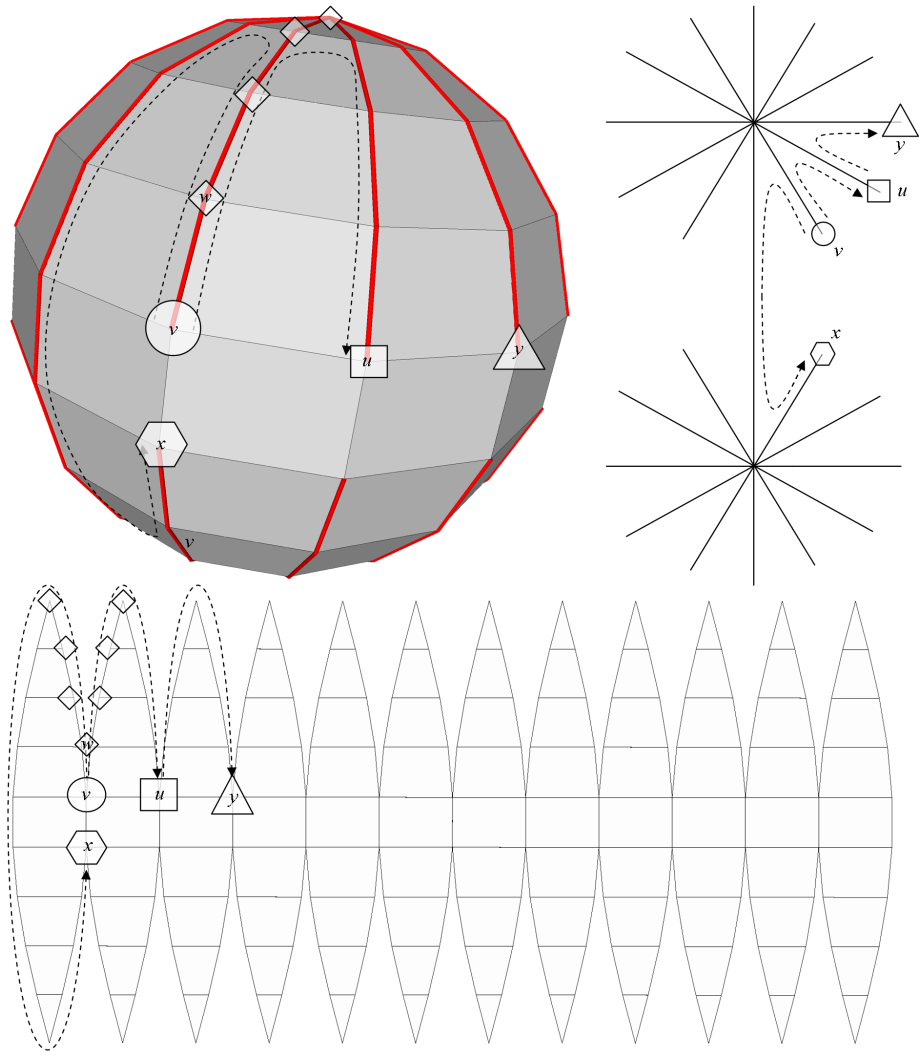


Figure 2.18: The sinusoidal projection unfolding of a sphere, showing the vertex v (circled) and its nearest neighbors which are leaf nodes in the cut-graph, u (square) and x (hexagon). The path $T_{v\to x}$ is the sequence of non-leaf vertices visited in the shortest counter-clockwise path from v to x; $T_{v\to u}$ proceeds similarly from v to u. $T_{v\to x}$ and $T_{v\to u}$ share a subset of vertices, indicated with diamonds. When unfolded, $T_{v\to x}$ unfolds to $\varrho_{x\to v}$ (note the re-ordering of terms to preserve direction) and $T_{v\to u}$ unfolds to $\varrho_{v\to u}$. The vertices shared between $T_{v\to x}$ and $T_{v\to u}$ are distinct in the unfolded paths.

edge were to be added to the cut-graph, then the disjoint section of geometry now separated from the surface would develop to a simple convex polygon in the plane. The same applies for u, the leaf node immediately before v in a counterclockwise traversal around the edges of the net.

2.6.2 Locally-convex cut-graph

Intuitively, in unfolding a cut, the right and left sides of the cut edges can only self-intersect if they approach each other. At the leaf of the cut, the second vertices of the left and right developments diverge by the angle deficit of the first (leaf) vertex; at all subsequent vertices, the left development can only turn further to the left, the right can only turn further to the right.

Theorem 2.9 *Any convex polyhedron which can be spanned by a locally-convex cut-graph can be unfolded without overlap.*

Proof. Consider u and v, two leaf-nodes of a locally-convex cut-graph T linked by a single uncut edge ($uv \notin T$) where there is a path $T_{v\to u} \in T$ which follows a clockwise traversal of the border edges of the unfolded net from v to u. This path contains no other leaf node of T (see Figure 2.18.)

In a clockwise traversal of the edges in the border of the unfolded net, label the last leaf node of T encountered before v as x and the next leaf node encountered after u as y. The path $x \to v \to u \to y \in T$ contains no other leaf nodes of T.

This path is now decomposed into its constituent arcs, which are given explicit direction. Label the right developments of $T_{x\to v}$, $T_{v\to u}$ and $T_{u\to y}$ as $\varrho_{x\to v}$, $\varrho_{v\to u}$ and $\varrho_{u\to y}$ respectively.

The structure of this proof is as follows: first, it will be shown that $\varrho_{v\to u}$ cannot self-intersect. Then it will be shown that there can be no intersection between $\varrho_{v\to u}$ and $\varrho_{x\to v}$, nor between $\varrho_{v\to u}$ and $\varrho_{u\to y}$. The operation of *collapsing a fork* is then introduced, and finally it is shown that $\varrho_{x\to v}$ cannot cross $\varrho_{u\to y}$.

Lemma 2.10 *The right development of the path $T_{v\to u}$ has no self-intersection when T is locally convex.*

Proof. By construction the edges of $T_{v\to u}$ are a subset of some closed convex curve on the surface. Given that a closed convex curve develops without self-intersection (Lemma 1.3, p. 15) it is clear that a subset of such a curve must also develop without intersection. □

Lemma 2.11 *Given vertices x, v and u on locally-convex cut-graph T as shown in Figure 2.18, then $\varrho_{v\to u}$, the right development of the path $T_{v\to u}$, will not intersect $\varrho_{x\to v}$, the right development of the path $T_{x\to v}$.*

Proof. Refer to subfigure Figure 2.19(a) on page 45.

Having established that $\varrho_{v\to u}$ and $\varrho_{x\to v}$ are convex in the unfolded plane, label as M_v the ray which proceeds from v and bisects the line segment $w^L w^R$: M_v is the perpendicular bisector of the right and left developments of w. It is immediately clear that M_v lies counterclockwise from the edge vw^R, which is tangent to $\varrho_{v\to u}$ at v. $\varrho_{v\to u}$ is a convex figure and therefore will never cross

M_v. In like manner the left development of $T_{v\to x}$ will also not cross M_v; the left development of $T_{v\to x}$ is another label for the right development of $T_{x\to v}$, and so the two arcs will never meet. □

However, it remains to be established that $\varrho_{x\to v}$ will not cross $\varrho_{u\to y}$. A case where this is conceivable is shown in Figure 2.19(a), where M_v and M_u actually cross. Referring again to Figure 2.19, define the operation of *collapsing a fork*:

Collapsing a fork Consider a *fork* in the cut-graph, i.e., a vertex w of valence ≥ 3 connected 'downstream' to some vertex r and 'upstream' to two branches of T which have exactly one leaf node each (labeled u and v). It has already been established that $\varrho_{r\to v} \cap \varrho_{v\to u}$ and $\varrho_{v\to u} \cap \varrho_{u\to r}$ are empty. Thus without loss of information the vertices u, v and w may be deleted from T and replaced with w', the intersection of the edges $r^L w^L$ and $r^R w^R$, which meet at an angle of $AD(w) + AD(v) + AD(u)$ radians. All other edges and vertices of the development remain unchanged.

Although none are shown in the figure, if there are any other (non-leaf) vertices in $T_{u\to w}$ or $T_{v\to w}$, they are also deleted and their angle deficits added to the angle between $r^L w^L$ and $r^R w^R$; the proof is otherwise unchanged. It should also be noted that the symmetry shown in figure 2.19 is shown only for clarity; the argument does not in any manner depend on symmetry.

One might wonder why the operation described is the collapsing of a fork, coalescing two leaves into one, instead of collapsing a single sub-branch of the fork (which would be a simpler operation and thus presumably more robust for the purposes of the proof.) However, in collapsing a single leaf-node, the angle incidence of one side of w–in the figure, the left-hand side of w, if v were collapsed alone–would exceed π. This would violate the requirements of local convexity; thus two leaves must always be collapsed together.

Lemma 2.12 *Given vertices x, v, u and y all leaf nodes of locally-convex cut-graph T as shown in Figure 2.18, the right development of $T_{x\to v}$ will not intersect the right development of the $T_{u\to y}$.*

Proof. Collapse the fork $w/v/u$ to find w'. The perpendicular bisector from w' separates the right development of $T_{w'\to y}$ from the right development of $T_{x\to w'}$, because both paths develop to a convex figure in the plane. □

It has now been shown that the development of a single path between leaves of a locally-convex tree cannot self-intersect; that the adjacent developments of two adjacent paths sharing one common end-leaf cannot intersect; and that given three sequentially-adjacent paths sharing two common leaf nodes that the middle path can be 'collapsed', allowing the proof to proceed on a figure with one fewer cut-graph branches and showing that the first branch cannot intersect the third. Therefore any convex polyhedron which can be spanned by a locally-convex cut-graph can be unfolded by that graph without overlap. □

2.6.3 Local convexity for trivalent polyhedra I

Observe that a trivalent vertex on a convex polyhedron cut at one edge or three edges will be locally convex. This could only fail to be the case if a face incident to the vertex had greater than π incident angle, an impossibility on a convex surface.

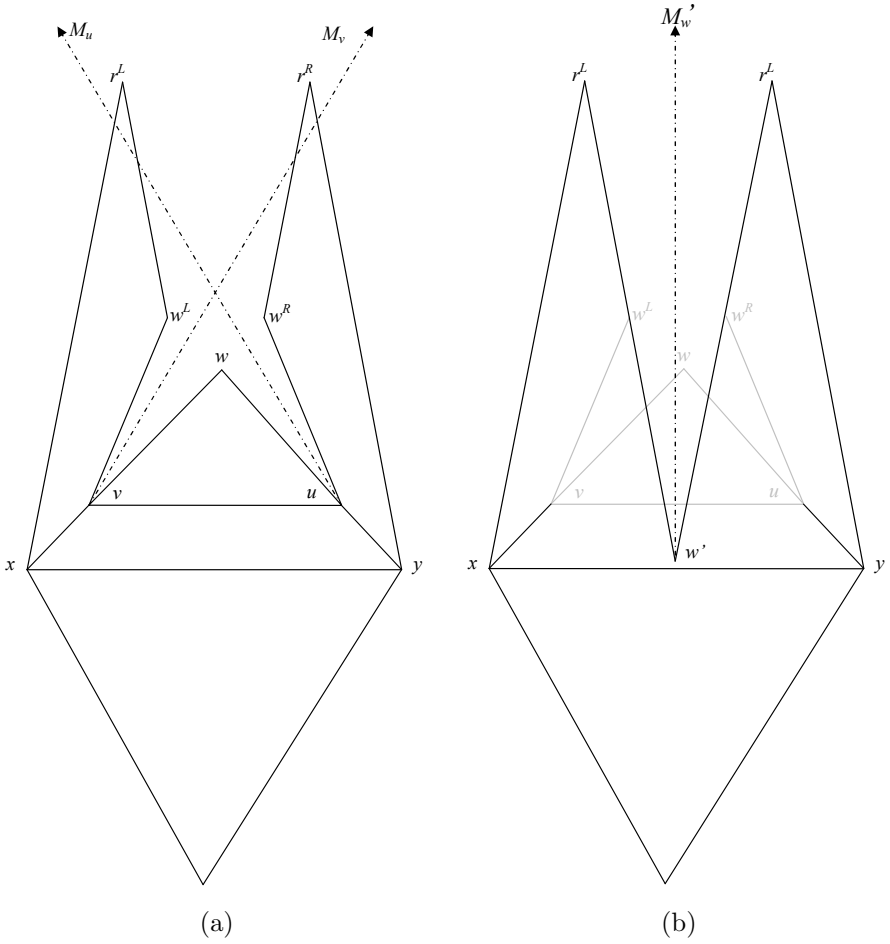


Figure 2.19: A fork is trimmed without disturbing the surrounding edges, showing that a branch of T may be collapsed without loss. Lines $w'r^L$ and $w'r^R$ meet at an angle of $AD(w) + AD(v) + AD(u)$.

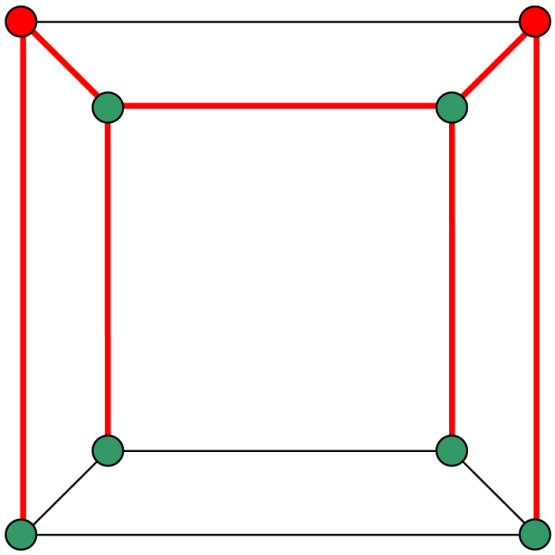


Figure 2.20: The one-skeleton of a cube is spanned by a trivalent graph but no spanning of the trivalent graph is a strictly binary tree. Here red edges (sides, back) are edges in the cut-graph and black edges (top front, bottom) are not. Green nodes show trivalent or monovalent corners, red nodes show bivalence.

A strictly binary tree is a connected acyclic graph of monovalent and trivalent vertices. (The term 'strictly' is used because it is a common convention in computer science to conside trees which have bivalent roots but are otherwise mono- or trivalent to be binary trees.) If it were to be shown that every trivalent convex polyhedron is spanned by a strictly binary tree then by Theorem 2.9 every trivalent convex polyhedron would be unfoldable.

By Steinitz (see Lemma 1.6, p. 18), a trivalent convex polyhedron P is spanned by a planar 3-connected graph G.

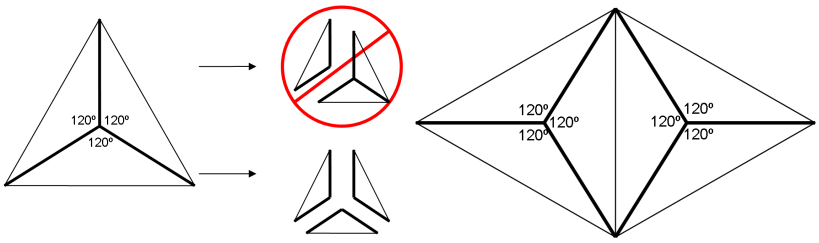
By the definition of a 3-connected graph, there exists a binary tree T, a subset of G, which spans P.

A binary tree has nodes of only degree one or three. Therefore every vertex of P is locally convex under T.

Then by Theorem 2.9 it would appear that T unfolds P without overlap.

Conjecture 2.13 *A strictly binary tree exists for every trivalent convex polyhedron.*

Rebuttal
The assertion, "there exists a binary tree which spans [a planar 3-connected graph]" is incorrect, for it fails to account for loops in the graph. The vertices of a cube cannot be spanned by a strictly binary tree (i.e., a loop-free graph with all vertices precisely monovalent or trivalent) (Figure 2.20.) □



(a) (b)

Figure 2.21: (a) A trivalent vertex which must be separated into triplets to preserve the conditions of local convexity (b) If two trivalent vertices have two 1-ring neighbors in common, local convexity would require that the four edges with shared endpoints be cut, creating an illegal loop in the cut-graph

2.6.4 Local convexity for the general case

Conjecture 2.14 *A locally-convex cut-graph can be constructed on any convex polyhedron.*

Rebuttal
A counterexample to the conjecture can be constructed.

If a vertex has three incident edges meeting at $2\pi/3$ (Figure 2.21(a)) then all three edges must be cut to generate a convex tree; leaving any edge uncut would give an incident angle of $4\pi/3$, which would exceed the convexity limit of π. Placing two such figures side-by-side (Figure 2.21(b)) and cutting all three edges at each vertex would cause a loop to be formed in the cut-graph, separating two triangles from the rest of the unfolding. □

2.6.5 Local convexity for trivalent polyhedra II

Conjecture 2.15 *A locally-convex tree exists for every trivalent convex polyhedron.*

Rebuttal
The cube supports no binary tree but it does still support locally-convex cut-graphs. (For examples, see Figure 2.1.) A slightly larger counterexample shows that not every trivalent convex surface has a 1-skeleton which can be spanned by a locally-convex cut-graph: the *regular dodecahedron* (Figure 2.22). The dodecahedron is trivalent and unfoldable with twelve sides and twenty vertices. The face angle of every face of the regular dodecahedron is $\frac{3}{5}\pi > \pi/2$. Every vertex must therefore be monovalent or trivalent on the cut-graph; a bivalent vertex would have a local turn angle of $\frac{6}{5}\pi$.

This dictates that the cut-graph must be a strictly binary tree, but the vertices of the dodecahedron cannot be spanned by a strictly binary tree (Figure 2.22(c).) □

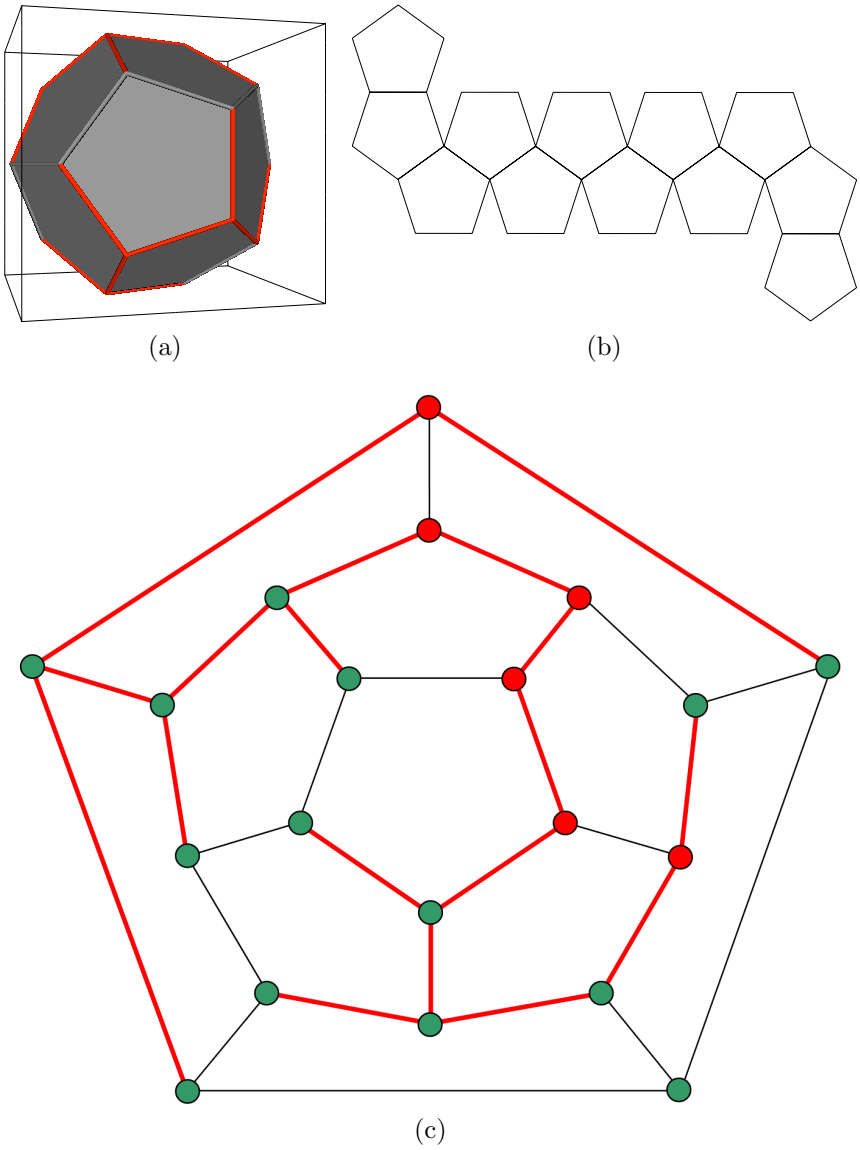


Figure 2.22: A dodecahedron, shown folded (a), unfolded (b), and in hyperbolic projection (c) to show the one-skeleton, with red bivalent vertices demonstrating the failure to construct a strictly binary tree. It is impossible to construct a locally-convex cut-graph on a dodecahedron.

2.6.6 The *domes* as a class of locally-convex polyhedra

Corollary 2.16 *Any convex polyhedron whose one-skeleton can be spanned by an acyclic tree which is monovalent or trivalent at all trivalent vertices of the polyhedron and locally convex at all multivalent vertices, can be unfolded without overlap.*

This demonstrates again the unfoldability of the domes.

Proof. This is a natural result of Theorem 2.9 and the observation, given above, that a trivalent vertex on a convex polyhedron cut at one edge or three edges will be locally convex. Trivalence is not required for local convexity; it suffices that the incident face angles to a vertex cut by multiple edges never add to greater than π between two cuts.

The *domes* are defined by construction to have one multivalent apex vertex a with every other vertex on the dome trivalent. The construction of a dome by truncation ensures that vertices are always created in pairs and every pair of vertices is adjacent to a higher trivalent vertex or a. Therefore the dome can be spanned by a tree which is strictly binary at every vertex except at a and which cuts every edge incident to a. Therefore all domes are unfoldable. □

2.6.7 O'Rourke's counterexample

Joseph O'Rourke has proposed a counterexample[2] to Theorem 2.9. The counterexample exploits an almost-flat vertex (a vertex with ϵ positive curvature) to create geometry which is effectively locally concave while still following the rules of local convexity. The model is shown above its (conflicted) unfolding in Figure 2.23(a); the unfolded net is shown alone in Figure 2.23(b). Figure 2.23(c) shows the net repaired, answering the question, "is this surface unfoldable at all?"; the answer is an easy yes.

Figure 2.23(d) shows the counterexample with each vertex painted to show local convexity. Green arcs show up to π incident radians between critical edges at each vertex; red arcs trace intervals which break the local convexity rules. Examining every vertex of the counterexample it is clear that the cut-graph is locally convex at every non-leaf vertex. However the top front vertex is *not* locally convex: it is a leaf node which does not comply with the leaf node rule that if the paths from the node to its first CW and CCW leaf neighbors were to be cut that the resulting polygon would be convex. As such the proposed counterexample does not actually meet the requirements of a locally convex cut-graph.

[2]Personal communications, 2007

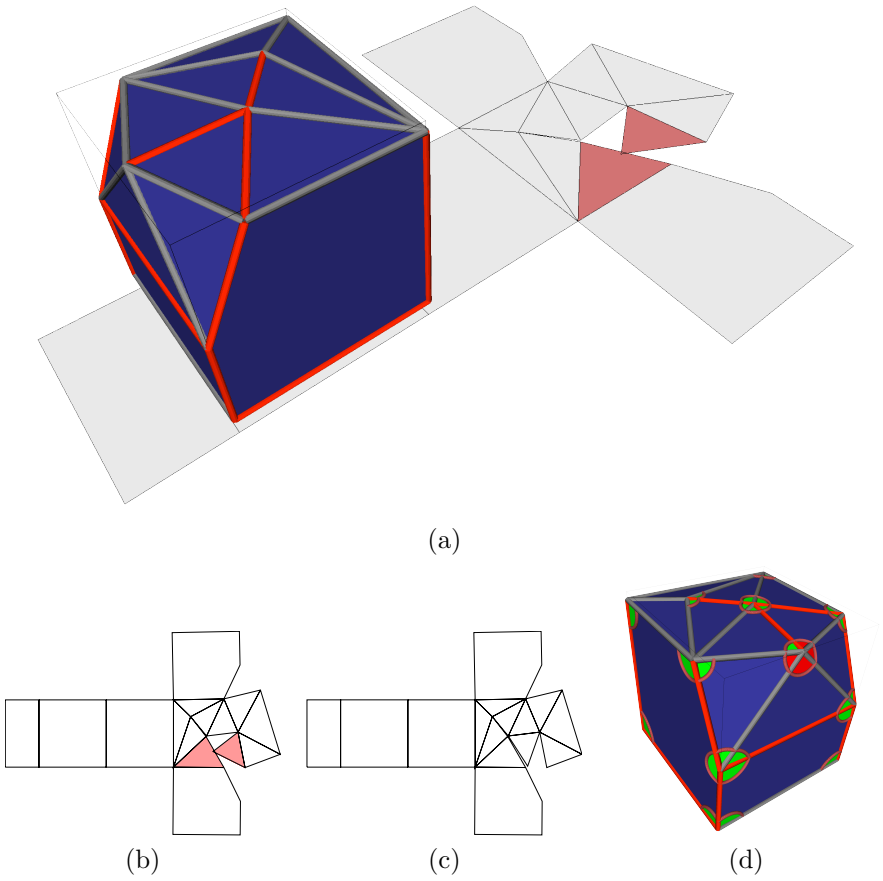


Figure 2.23: (a) O'Rourke's original counterexample to the Local Convexity Theorem (slightly modified for convexity): the cut-graph given appears to be locally convex, but the unfolding self-intersects. (b) An alternate unfolded net, free of overlap. (c) The overlap is fixed by a single edge-swap. (d) The proposed counterexample rendered with local convexity highlighting. Green arcs show vertices which are locally convex; red arcs show vertices which are not. Note the vertex at the front of the top face, a leaf-node which fails its local convexity test because the edge joining it to its first leaf-node counterclockwise neighbor is not on the convex hull of the set of cut-graph edges which link them.

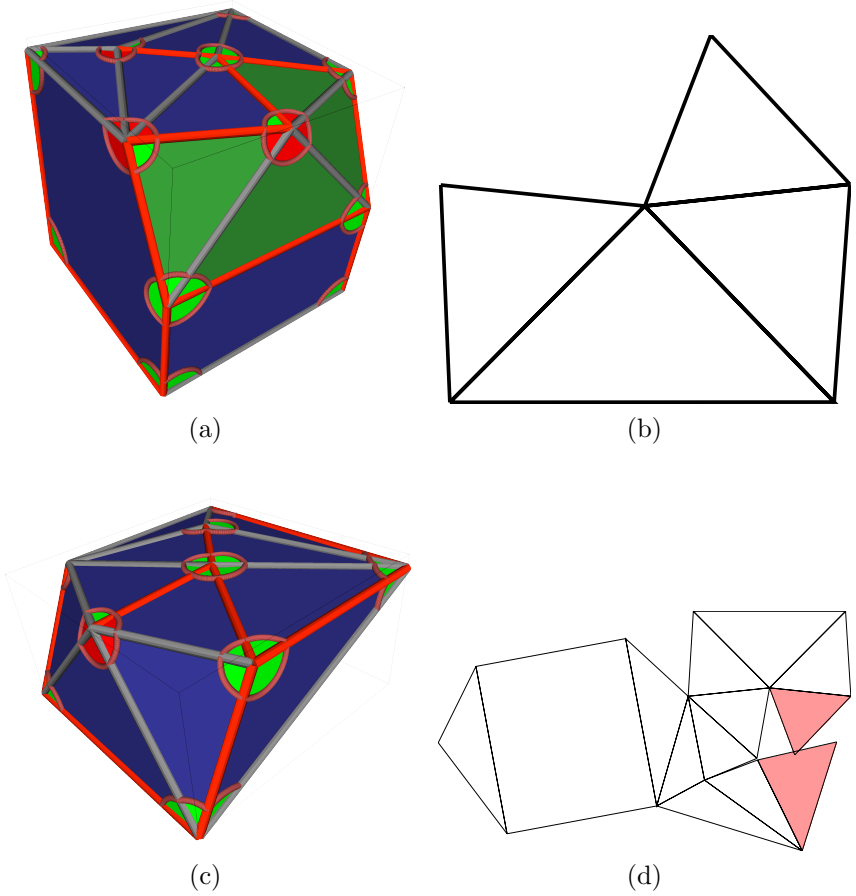


Figure 2.24: The O'Rourke counterexample, edited: the edge between the non-locally-convex leaf-node and its neighbor has been cut to highlight the inapplicability of the counterexample. (a) The section which being detached from the main unfolding is shaded light green. (b) The (clearly nonconvex) isolated region. (c), (d) The model as given contains extraneous faces and may be simplified to a thirteen-faced convex cap. The simplified model is useful in analysis in that it has fewer extraneous (non-overlapped) unfoldings: it has 26,418 possible nets, of which 23,586 have no self-intersection (90.30%.)

2.7 The Anchorable Convex Hull

This section presents steps toward an iterative proof, in which an unfolding is assembled face-by-face in a partial ordering which generates a series of stable states each of which can then provably be advanced to the next stable stage. This approach is motivated by the observation that many unfolding methods produce unfoldings which often show clear concentric rings in their layout; consider the unfolding of a sphere shown in Figure 2.25. A series of rings centered on the heart of the unfolding shows that concentric rings of faces are generated during development.

The ideas first seen here are not carried to a complete proof. Instead they inspire directions of further research, which are explored in Chapter 3.

Recall that a *valid partial unfolding* is a connected unfolded subset of the faces of a polyhedron which is known not to self-intersect (p.11.)

Conjecture 2.17 *There exists a set of rules by which a valid partial unfolding may be extended while preserving its developability.*

Supporting Arguments
The goal will be to describe precisely a particular form of valid partial unfolding, a subset of the polyhedron whose convex hull in the unfolding plane would have certain 'desirable' traits. This well-behaved subset could then be evolved through a series of face-gluing operations into the next larger well-behaved subset, preserving those same attributes. This approach depends on finding a nested series of convex hulls within the unfolding. Each such hull is the unfolding of a *closed convex curve*, i.e., a curve C where $c_0 = c_{n-1}$ and $\lambda(c_i) \leq \pi \forall c_i \in C$ (see p. 4.) (This should not be confused with the *locally convex tree* discussed in Section 2.6, where both $\lambda(c_i)$ and $\rho(c_i) \leq \pi$.)

Define an *Anchorable Convex Hull* ("ACH") (Figure 2.25) as a convex planar polygon with the following traits:

- The Anchorable Convex Hull is the planar convex hull of a valid partial unfolding of the polyhedron.
- Certain edges of the Anchorable Convex Hull are *anchorages* (Figure 2.25(a).) An anchorage is an edge of the ACH defined by a single face within the partial unfolding where that face lies with an edge against the convex hull. Informally, an anchorage is where another face could be 'glued' to the outside of the ACH. For every anchorage there is exactly one face in the polyhedron which could be glued to that anchorage.
- Each anchorage shares a vertex in the original polyhedron with the next and previous anchorages (where 'next' and 'previous' refer to 'the anchorage encountered next/previously when visiting edges around the Hull in a clockwise ordering) and the path of anchorage edges forms a single loop on the polyhedron. The connected path of anchorages forms a closed convex curve on the surface.
- After unfolding, neighboring anchorages may lie some distance apart in the plane, separated spatially and also separated in the Hull by one or more non-anchorage edges. Those edges of the ACH which are not anchorages are called *fens* (Figure 2.25(a), (b).) As an unfolding blossoms outwards

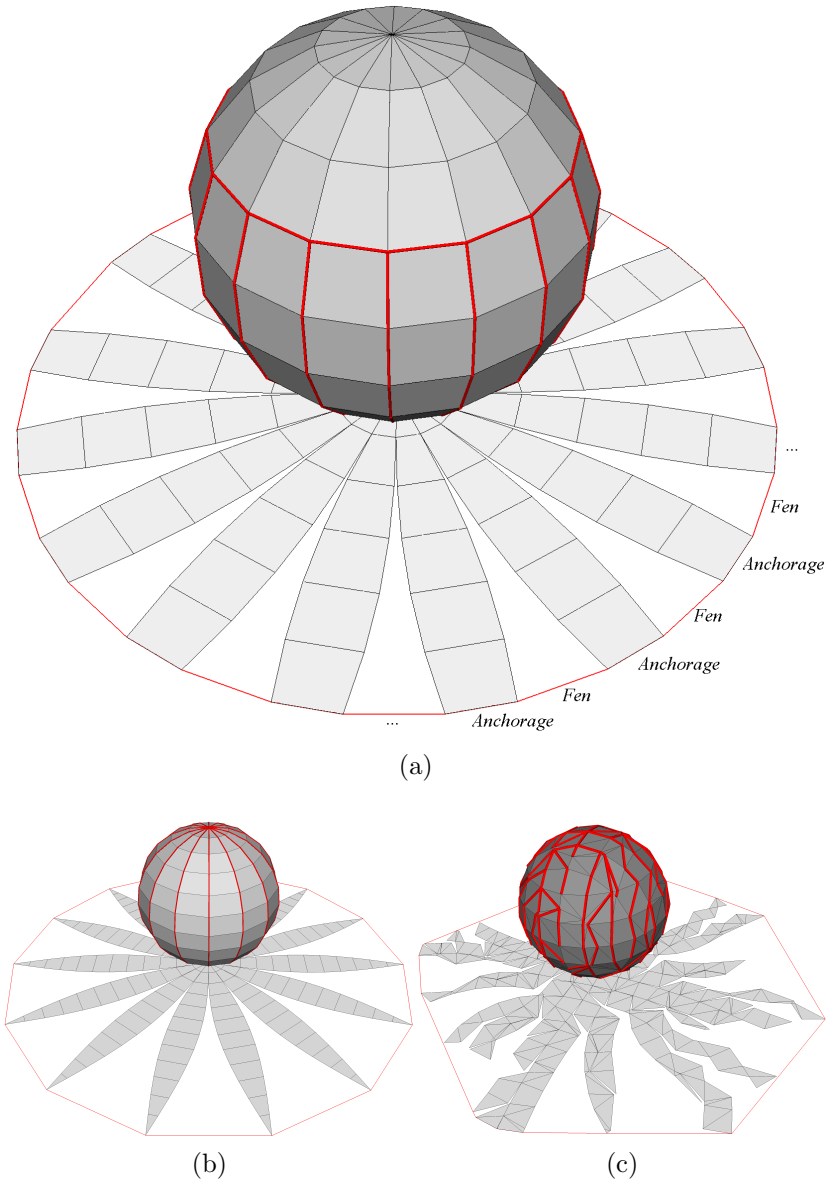


Figure 2.25: (a) An Anchorable Convex Hull showing anchorages and fens for the partial unfolding of a parametrically-generated sphere (b) The finished unfolding is now entirely fens (c) Unfolding of a sphere modeled with marching cubes shows irregular polygonalization, leading to an Anchorable Convex Hull mixing anchorages and fens with multiple fens lying adjacent along some stretches of the hull.

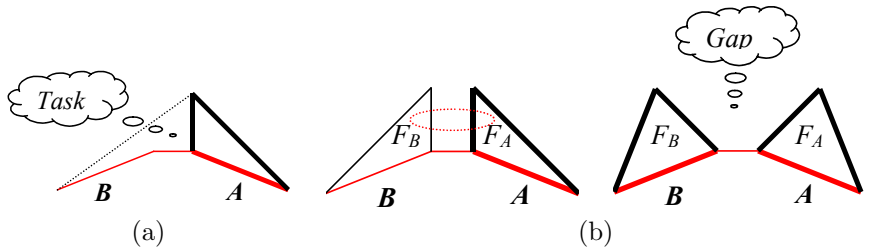


Figure 2.26: (a) Adding a face creates a task (b) If the two new faces do not share an edge then a gap is created

from its center, the unfolded faces will crack along cut edges and spread apart. Each such crack ends in a fen edge of the Anchorable Convex Hull. Each fen edge caps the opening of a unique branch of the growing cut-graph. Multiple fens may lie between anchorages (Figure 2.25(c).)

- There is no face in the polyhedron which is not inside the ACH which shares an edge with any face within the ACH that is not an anchorage edge. Every face which can be glued to the ACH at any internal edge, must already be so glued.

If the unfolding is not yet complete then each branch of the growing cut-graph remains distinct and bounded by a fen. If the partial unfolding contains every face of the polyhedron–that is to say, if it is actually a full unfolding–then the union of the set of all edges bounded by each fen with the set of all anchorages is the complete cut-graph.

By construction, the angle between any two adjacent anchorages is at most π. This is reminiscent of the construction of the locally-convex cut-graph (Section 2.6,) which depended on a similar design, but on the Anchorable Convex Hull it applies to the uncut edges of the unfolding and travels across cuts instead of along them.

The angle between two anchorages A and B which share a common endpoint v on the polyhedron but which are separated by one or more fens in the unfolding is exactly the sum of the incident angles of the faces at v which lie within the Hull, minus the total angle deficit of each branch of the cut-graph bounded by each intervening fen. A and B can only be separated by fens, never by another anchorage, for that would imply an intervening edge on the original polyhedron and the anchorages are defined as a connected path.

The angle between two anchorages A and B which lie immediately adjacent on the Hull, sharing some common vertex v, is exactly the sum of the incident angles of the faces incident to v which lie within the Hull on the polyhedron. The Hull is known by construction to be convex, therefore the total incident angle within the hull at v is $\leq \pi$. (Were it more than π, v would be concave.)

2.7.1 Anchorable Convex Hull - walkthrough

A listing of the algorithm of the Anchorable Convex Hull can be found in Algorithm 1, p. 55.

Algorithm 1: The Anchorable Convex Hull

$H \leftarrow < A, F >$ is an Anchorable Convex Hull of P
  $A \leftarrow$ the set of anchorages in H
  $F \leftarrow$ the set of fens in H
$Tasks \leftarrow \emptyset$ will be a list of Tasks, sorted by distance in number of edges from H, closest edge first
$a \leftarrow$ the first anchorage in A
`Glue`(f_a, a)

`Glue`(f, a)
$H' \leftarrow H \cup f_a$
$A' \leftarrow A - a$
for each edge e of f_a **do**
  if $e \notin A'$ **then**
    Insert e into $Tasks$, preserving ordering
  end if
end for
while $Tasks \neq \emptyset$ **do**
  $t \leftarrow$ `Pop`$(Tasks)$
  Find a_e, the next open anchorage beyond t
  `Glue`(f_{a_t}, a_t)
end while

The heart of the construction of an Anchorable Convex Hull is the act of 'gluing' one or more faces to the Hull until the resulting figure is once again compliant with the rules above. In the assembly of an ACH two construction operations are used:

- *Tasks* form the core of an advancing-front algorithm for ACH construction (Figure 2.26(a).)
- *Gaps* may form between Tasks, inducing more complex goals which will require special resolution (Figure 2.26(b).)

The algorithm begins with a valid Anchorable Convex Hull: any face of the polyhedron. Each face of a convex polyhedron must itself be convex, yielding a viable initial convex hull which happens to have no fens. (That said, a cursory inspection of real surfaces makes it clear that some faces are much more amenable to being the heart of a series of rings than others. The careful choice of seed face is worthy of further research.)

By construction, all anchorages lie on a convex hull. This guarantees that the operation of gluing a face to an anchorage does not introduce a conflict in the unfolding. In fact, the ACH construction ensures that faces could be glued to every anchorage on the ACH and none of them would conflict, with each other or with any face within the ACH. Consider two anchorages A and B which meet on the polyhedron at a shared vertex v and two faces F_A and F_B to be 'glued' to A and B respectively. Noting that F_A and F_B must both be incident to v:

- Where A and B unfold to lie adjacent in the Hull, the sum of the incident angles of the faces meeting at v in the plane can at most equal the sum of the incident angles of the faces meeting at v on the polyhedron; given that $AD(V) \geq 0$, there can be no intersection between F_A and F_B. The two faces are separated by at least $AD(V)$ radians of arc.
- Where A and B unfold to positions on the Hull which are separated by one or more fens, recall that the lines containing the line segments A and B meet in the plane at an angle which is the sum of the incident angles of the faces incident to v, minus the total angle deficit of the cut-graph branches bounded by each of the intervening fens. The angle of encounter plus the face angles of F_A and F_B can thus never exceed 2π. Therefore F_A and F_B cannot cross.

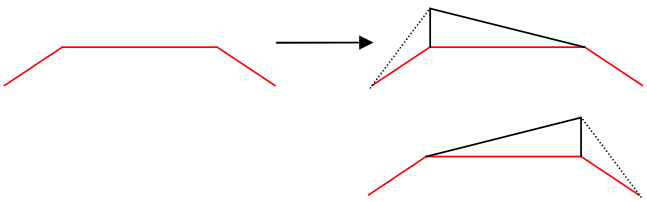
Thus the unfoldings of F_A and F_B can never overlap.

By construction any Anchorable Convex Hull contains a valid unfolding. It remains only to be shown that there exists a set of rules which can always evolve one ACH into another ACH; that will prove that an ACH can be constructed for the complete unfolding of any convex surface, thereby showing that any suitable surface can be unfolded.

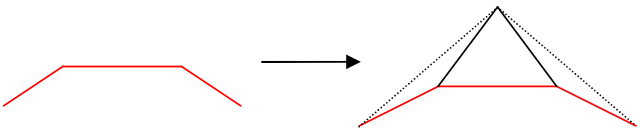
Consider the addition of a single face to an anchorage. This will create one of three scenarios (up to symmetry):

1. 

   The anchorage edge is replaced by two new anchorages. No new tasks are created. The result is a new valid ACH, guaranteed to contain no conflicts.

2. 

   One edge of the new triangle lies on the outline of the convex hull but the other edge lies within it. The anchorage edge is replaced by one new anchorage and one new task. The task's goal will be to fill in the gap created to one side of the anchorage; counter-clockwise in the upper illustration, clockwise in the lower.

3. 

   The opposing vertex of the new triangle extends the convex hull but neither opposing edge of the triangle lies on the convex hull. The anchorage edge is replaced by two new tasks. The goal of each task will be to fill in the gaps to either side of the former anchorage.

This introduces the concept of a 'task' (Figure 2.26(a).) A *task* is a job which must be completed through a series of gluing operations which will yield

a new, valid ACH. A task is created when a face is glued to the unfolded net which extends the convex hull but adds an internal edge, within the bounds of the ACH and not on its border, to which another face which has not yet been glued on might be added which would fall within, or across, the border of the ACH. Such a gap must be repaired before the figure can once again be called an ACH.

Tasks are oriented. A given task seeks to fill a gap in the convex hull in either the clockwise (CW) or counter-clockwise (CCW) directions.

Consider a pair of neighboring anchorages A and B in an Anchorable Convex Hull, where A lies clockwise from B. Examine the case where gluing a face F_A to A has lifted the convex hull beyond the edge B, creating a CCW task to be resolved (Figure 2.26(a).)

It is known that the anchorage B is free; F_B can safely be glued to B without fear of introducing a conflict. The three possible scenarios hold once more and the new face could create a new CCW task. If F_B creates no new task then this task is done; if F_B creates a new CCW task, it can be resolved recursively.

It is possible that the CW edge of F_B is not the CCW edge of F_A, even though they share the common vertex v (Figure 2.26(b).) If so then a 'gap' has been introduced in the border of the convex hull. A *gap* is a space along the ACH between two filled anchorages. Gaps are a new kind of task.

Recursive resolution of a task can be seen informally as a task 'traveling' around the outer edge of the convex hull. At each new anchorage, the task either terminates or carries on; perhaps leaving gaps in its wake. As a task is propagated around the hull, it will always be able to advance because it is known that the hull's anchorages are waiting, until the task has wrapped around the convex hull and approaches itself from the other side. As the task reaches itself from the other side of the ring, it becomes a gap.

Each gap must now be filled across some vertex v with the fan of faces which lie between F_B and F_A and which have v as one corner. This operation is guaranteed to not induce conflicts, because v has positive angle deficit. It will suffice to ensure that the new figure complies with the other rules of the ACH.

Although the algorithm described above is illustrated with triangular faces, it extends easily to nonsimplicial polygons. As each face is glued to its anchorage and its edges are evaluated to (potentially) become tasks, each edge is processed independently, beginning with the two edges closest to the anchorage and ending with the furthest edge. By design, each edge can lead to one or more tasks which will 'fill in' the ACH up to the level of the next edge in the face.

Figure 2.27 shows two examples of convex polyhedra which are well-suited to unfolding by the rules of the Anchorable Convex Hull. A sphere generated by marching cubes is circumscribed by a closed convex curve at the boundary plane of every cube (Figure 2.27(a)) and a convex polyhedron parametrically generated from a bivariate field will have as many closed convex curves along one axis as its model has resolution on that axis (Figure 2.27(b).)

Rebuttal

Unfortunately, it is difficult to prove that the face fan at v conforms to the rules of the Anchorable Convex Hull. Gaps can develop into 'complex gaps', in which multiple faces fill a gap, perhaps crossing the boundary of the ACH, perhaps embedded within it (Figure 2.28(a).)

This problem of resolving a gap is much more difficult than the problem of resolving a task. With tasks the ACH guaranteed certain preconditions which

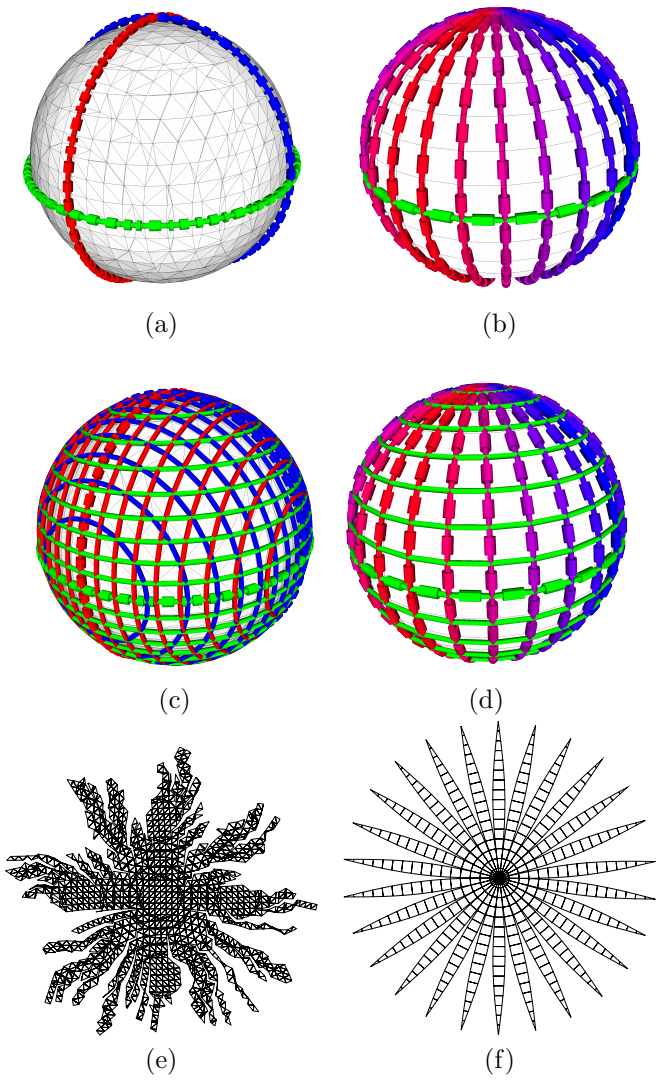
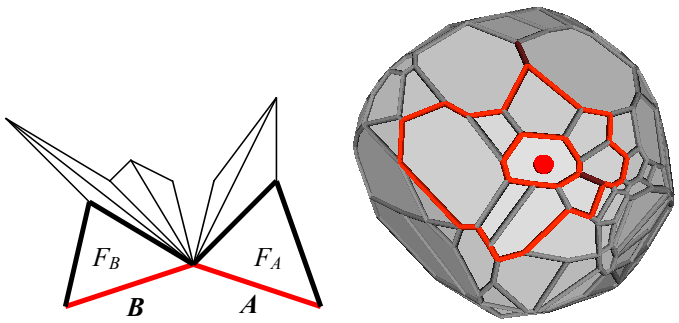


Figure 2.27: (a) A triangulated implicit-surface sphere with the three unique locally-convex geodesic loops highlighted. (b) A parametrically-generated sphere computed from an $n \times m$ grid showing n vertical locally-convex geodesic loops and one horizontal loop at the $m/2$ ring. (c) The implicit sphere with added rings showing concentric closed convex curves. The concentric rings show six suitable starting points for the Anchorable Convex Hull algorithm. (d) The parametric sphere with concentric closed convex curves. Note that in contrast to the implicit sphere, here only one axis and two suitable starting points are concentric. (e) The ACH unfolding of the implicit sphere. (f) The ACH unfolding of the parametric sphere.



(a) (b)

Figure 2.28: (a) A gap whose internal faces will be difficult to fit to the rules of the ACH (b) Most random polyhedra have no closed convex curve larger than a few faces.

ensured resolution. When attempting to resolve gaps there seem to be no known conditions which guarantee that a gap will be 'well behaved', i.e., resolvable by a recursive division of its component triangles. While not a rebuttal to the conjecture *per se*, this difficulty in the algorithm has been the primary reason that exploration of the Anchorable Convex Hull has not been carried further.

The stronger rebuttal to the conjecture that the ACH rules can extend any partial unfolding is that the rules would seem to have very limited applicability in the general case. A randomly-generated polyhedron has very few closed convex curves larger than two faces, which diminishes the general applicability of the Anchorable Convex Hull method dramatically (Figure 2.28(b).) Other common counterexamples, such as the icosahedron and the class of banded polyhedra (see Section 2.8) also do not contain closed convex curves. □

2.7.2 Insights from the Anchorable Convex Hull

The Anchorable Convex Hull would seem to show that any convex surface which contains multiple nested closed convex curves is always unfoldable. The proof has not been made with mathematical rigor, as too many counterexamples are known to which this method cannot apply and the author felt that other avenues of exploration would bear more fruit. To characterize the class of polyhedra to which these rules apply may be quite difficult.

Still, there is potential in this method, and the ideas first developed here appear again in the author's later research. The vertex ordering which is implicit in the ACH is made explicit in the alpha-beta rules (Section 3.3.) The concentric layout of the net of an ACH unfolding is very similar to that of a Least Height Unfolder (Section 4.5.3.) While the Anchorable Convex Hull may be unduly limited in application or extent, it has served as an excellent incubation space from which valuable ideas have emerged.

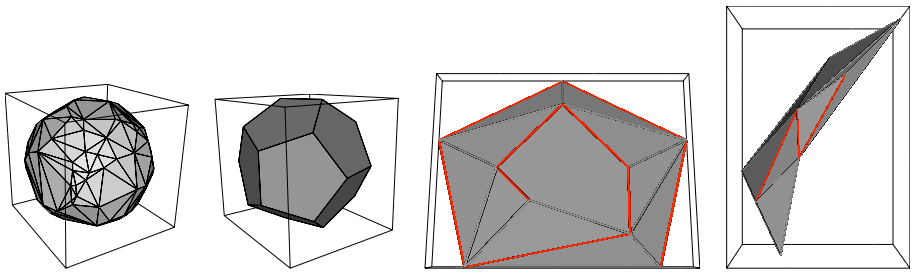


Figure 2.29: Each of these models is a counterexample to a particular unfolder or conjecture.

2.8 Polyhedral Banding: Building Counterexamples

Several local counterexamples have been found in the quest to prove that all convex polyhedra are developable. Each counterexample is a surface, often paired with a particular cut-graph, which contradicts a specific scenario or presents a surface to which a specific method cannot apply. This is not to say that these surfaces are not developable: every surface so described can be developed, but not by every method known. A few of these counterexamples, shown in Figure 2.29, include:

- A random convex polyhedron, which will often defeat many schemes which implicitly exploit regularity, such as the Anchorable Convex Hull (Section 2.7), and a few that would seem not to, such as the discrete Star Unfolding (Section 2.3)

- The dodecahedron, which cannot be unfolded to a locally-convex cut-graph, putting it beyond the purview of Local Convexity (Section 2.6) or Cut-Tree Truncation (Section 2.4)

- The spiral-cut that does not encircle the model, which showed that fixing $\lambda(p) \leq \pi$ could turn a cut-graph too far inwards upon itself and thereby defeated the Open Convex Curves conjecture (Section 2.5)

- The convex cap, which demonstrates that a trivalent vertex cannot always be truncated while preserving unfoldability, another rebuttal to the applicability of Cut-Tree Truncation (Section 2.4, p. 38)

Random convex polyhedra and the dodecahedron are, in a sense, only symptomatic; they serve as counterexamples to methods, not to particular cases, and many other methods exist which can unfold them. The second two examples are more interesting: here are particular models which, when cut a particular way, cannot be unfolded. In short, it is possible to devise a polyhedron which cannot be cut in certain combinations; can one be constructed which cannot be cut at all?

2.8.1 Polyhedral banding

In (O'R07), Joseph O'Rourke introduced a polyhedral surface which was intended as a counterexample to the conjecture put forth in (ADL$^+$04) that all prismatoids could be 'band-unfolded', that is, unfolded by opening the strip of faces between top and bottom of the prismatoid into a single connected band. The construction of O'Rourke's counterexample inspires a new technique for the construction of polyhedra which are demonstrably 'difficult to unfold': convex polyhedra for which it can be shown *a priori* that a non-zero fraction of cut-graphs must unfold the polyhedra to overlap.

The method used to construct these 'difficult to unfold' polyhedra is called *polyhedral banding*. Banding modifies a source polyhedron P_0 by replacing triangular faces with harder-to-unfold hex assemblies which, when unfolded into the plane, take up more space than the original triangle. An algorithm for the construction of a banded polyhedron is given in Algorithm 2 on page 64.

O'Rourke's original polyhedral banding model is shown in Figure 2.30. It is a closed convex polyhedron with eight sides: one smaller hexagon on top, six quadrilaterals and one larger hexagon on the base. The model shares the interesting trait with all polyhedrally banded models, that if all but one of the edges of the base hexagon (u_iu_{i+1}) are cut and only one of the edges from the base hexagon to the inner hexagon (u_iw_i) is cut then overlap is inevitable. In fact, 27.80% of the possible unfoldings of O'Rourke's polyhedron self-intersect.

If O'Rourke's banded surface is converted to a convex cap by deleting the base hexagon, the percentage of unfoldings with overlap for a single isolated 'hex assembly' climbs to 47.81%.

The unfoldings of the hex assembly occupy a region in the unfolded plane with larger surface area (that is, their convex hull has larger area) than the triangle which the hex assembly replaced, because there are gaps between the cut edges. This raises an intriguing possibility: what would happen if several of these caps were placed closely together? Could they be assembled in such a manner that their unfoldings were forced to overlap, making self-intersection inevitable in every unfolding?

The construction is shown for a tetrahedron in Figure 2.31, a banded surface with 28 faces to the original tetrahedron's four. Interestingly, the banded tetrahedron turns out to be a counterexample to the Least Height Unfolder (Section 4.5.3), an unfolding algorithm which had performed exceptionally well when tested against randomly-generated convex polyhedra. That said, the banded tetrahedron is still unfoldable. The overlap-free development shown in the figure was computed using the Collision Repair algorithm (Section 3.4.)

As an example in support of the claim that banded polyhedra are more 'difficult' to unfold, the probability of a randomly-generated simplicial polyhedron with 25 to 30 faces being unfolded by a single randomly-generated spanning tree of the faces is approximately 85.53% (determined by Monte-Carlo simulation.) In contrast the probability of a randomly-generated spanning of the 28 faces of the banded tetrahedron unfolding without overlap is approximately 9.12%.

2.8.2 The polyhedral banding algorithm

The author's algorithm for construction of a banded surface is given in Algorithm 2 (see also Figure 2.30(a).) The design of the algorithm is as follows:

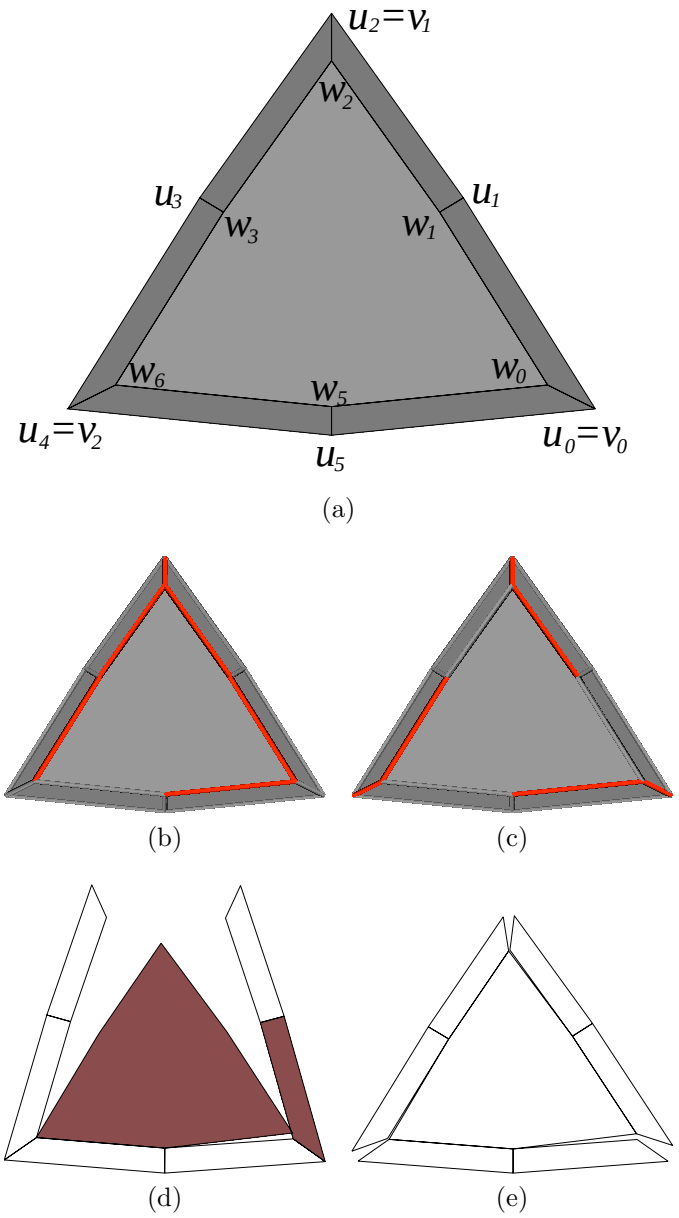
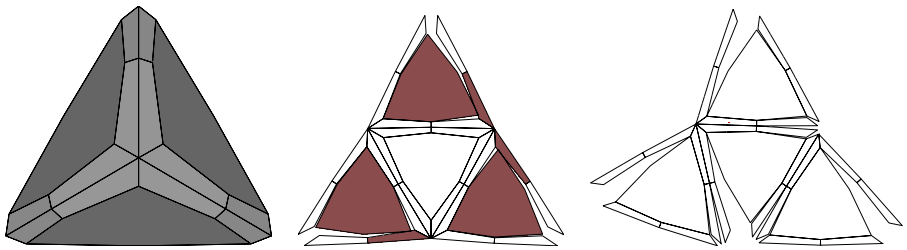


Figure 2.30: (a) O'Rourke's counterexample to the conjecture that all prismatoids can be unfolded by opening the strip of faces between top and bottom into a single connected band (ADL+04). There is no position of the top face which does not clash with the band faces (b, d) unless the band is cut more than once (c, e). The model uses the banding technique to reduce the number of possible unfoldings.



(a) (b) (c)

Figure 2.31: (a) The banding technique applied to the triangular faces of the tetrahedron. This surface is a strong counterexample for the Least Height Unfolder (Section 4.5.3,) which unfolds to overlap as shown in (b) but the surface is still unfoldable; roughly 12.42% of its unfoldings do not self-intersect (percentage found by Monte-Carlo simulation.) The overlap-free unfolding shown in (c) was found using the Collision Repair algorithm (Section 3.4.)

A new vertex u_{2i+1} is calculated from the midpoint of every edge $u_{2i} \rightarrow u_{2i+2}$. The new vertex lies somewhere between the exact midpoint of the edge and the intersection of the projection of the midpoint onto the sphere whose radius is the average radius of the vertices of the adjacent faces. The precise position of the new vertex between these two choices is set by the constant *dentEpsilon*, which specifies how far out the 'denting' of each edge goes, measured from zero (no dent, radius is radius of the midpoint of the edge) to one (dent out to the same radius as the radii of the endpoints of the edge).

At each new vertex u_{2i+1} a tangent plane is calculated from the average of the normals of the faces to either side of the original edge. The convexity of P_0 ensures that these tangent planes will do not intersect the polyhedron.

For each face f in P_0, the normal to f is intersected with the tangent planes associated with each of the edges of f. Again, all such intercepts must lie outside the polyhedron. The closest such intercept point is labeled as the *tip* of the virtual pyramid to construct upon face f. The tip is then lowered a fraction toward the center of the new face, by the constant *tipDescent*. *tipDescent* determines how far down the tip of the pyramid is depressed below its maximum, chosen from zero (tip is exactly at the lowest intersection of the tangent planes, i.e., new quadrilaterals on either side of the original edges will be coplanar) to one (tip is embedded in the plane of the out-dented vertices, i.e., a degenerate pyramid).

Having now computed a shallow virtual pyramid above every face, with twice as many edges around the pyramid as the original face, the pyramid is truncated to generate the quadrilaterals and inset hexagon. (Or octagon, or appropriate $2n-$gon for an original face with n sides.) Truncation is by the 'slice plane', a plane with normal equal to the normal of the face and passing through a point positioned at the constant fraction *slice* between the center of the new face and the pyramid's tip. The constant value *slice* is chosen from zero (cut at the height of the out-dented vertices) to one (cut at the tip of the virtual pyramid, i.e., do nothing).

The output from each face of P_0 is one new $2n-$gon and $2n$ quadrilaterals.

Algorithm 2: Polyhedral Banding

$P_0 \leftarrow$ a convex polyhedron, vertices ordered counterclockwise by face
$P_1 \leftarrow$ the output, a banded convex polyhedron
$C \leftarrow$ the center of P_0
Define constants: $dentEpsilon \leftarrow 0.1$; $tipDescent \leftarrow 0.5$; $slice \leftarrow 0.12$
for all Face $f \in P_0$ **do**
  $N \leftarrow$ the normal of f
  $R \leftarrow$ the ray from the center of f with direction N
  $len_A \leftarrow (\sum_{i=0}^{i=n-1} \|v - C\|)/n$
  for all $v_i \in f$ **do**
    $mid \leftarrow (v_i + v_{i+1})/2$
    $len_B \leftarrow \|mid - C\|$
    $u_{2i} \leftarrow v_i$
    $u_{2i+1} \leftarrow (len_B + dentEpsilon * (len_A - len_B)) * ((mid - C)/len_B) + C$
    $f_i \leftarrow$ the face neighbor to f at edge $v_i v_{i+1}$
    $p_i \leftarrow$ the plane passing through u_{2i+1} with normal the average of the normals of f and f_i
    $tip_i \leftarrow R \cap p_i$
  end for
  $c_f \leftarrow (\sum_{i=0}^{i=n-1} u_{2i+1})/n$
  $tip \leftarrow tip_k$ closest to c_f
  $subtip \leftarrow c_f + tipDescent * (tip - c_f)$
  $cutpoint \leftarrow subtip + slice * (subtip - c_f)$
  $p \leftarrow$ the plane through $cutpoint$ with normal N
  for all $v_i \in f$ **do**
    Let r_{2i} be the ray from u_{2i} toward $subtip$
    Let r_{2i+1} be the ray from u_{2i+1} toward $subtip$
    Let $w_{2i} \leftarrow p \cap r_{2i}$
    Let $w_{2i+1} \leftarrow p \cap r_{2i+1}$
    Add two new quadrilaterals to P_1 with vertices $[u_{2i}, u_{2i+1}, w_{2i+1}, w_{2i}]$ and $[u_{2i+1}, u_{2i+2}, w_{2i+2}, w_{2i+1}]$
  end for
  Add a new face to P_1 with vertices $[w_0, w_1, \ldots, w_{2n-1}]$
end for

The $2n-$gon is clearly planar, because its border is the set of edge-intercepts between the virtual pyramid and the slice plane. The quadrilaterals are also planar, because each quadrilateral is the truncation by the slice plane of the virtual triangle $\triangle\ u_{2i}$, u_{2i+1}, *subtip* or $\triangle\ u_{2i+1}$, u_{2i+2}, *subtip*.

The output polyhedron P_1 is guaranteed to remain convex, because the virtual pyramids with tips at *subtip* were all chosen to be shallower, by the fraction *tipDescent*, than the nearest intersections of the tangent planes to the surface above the face.

Note that the values for *dentEpsilon*, *tipDescent* and *slice* reflect an assumption that P_0 is of roughly unit radius; these values were found through experiment to produce useful results on the particular test cases chosen. The interested reader may wish to explore alternative constants, as these are quite specific to the original test data (primarily unit-radius tetrahedra and icosahedra.)

2.8.3 A class of convex polyhedra with few edge unfoldings

The short note *A Class of Convex Polyhedra with Few Edge Unfoldings*[3](BO08) builds on the idea of polyhedral banding to define the *banded geodesic spheres*, a class of polyhedra which are provably 'difficult' to unfold. These polyhedra are constructed by the recursive subdivision, to some level L, of an icosahedron, and then polyhedrally-banding the geodesic approximation of a sphere that results. Figures 2.32 and 2.33 show the first levels of banded geodesic sphere, $L = 0, 1, 2, 3$.

(BO08, p.6) introduces the concept of *hexagon overlap*, in which a hex assembly on a surface has been unfolded to intersection between its component faces. If the faces of the hex assembly are still a *locally-connected* net after cutting (that is to say, if there is no face in the hex assembly such that the only path to the disconnected face from the central hex is through polygons not belonging to the hex assembly) then the circumstances in which overlap is guaranteed to occur may be clearly delineated. Referring once again to Figure 2.30(a), in which the vertices w_i are the border of the inner hexagon and the vertices u_i border the outer perimeter of the hex assembly, the minimal conditions that lead to hexagon overlap are: exactly one edge w_iw_{i+1} is not cut, and exactly one edge u_iw_i is cut.

Given that there are six edges w_iw_{i+1} and six edges u_iw_i in a hex assembly, it is deduced that there are 6^2 possible nets of the faces of the hex assembly

[3]The following passage appears in the Collaboration Statement which accompanies this dissertation:

> Joe and I put this paper together when we found that we'd both been working on his banded counterexample and had each produced a banded icosahedron independently. The paper was built on an earlier short note that Joe had written; my contribution begins at page 5, where I wrote software to build the banded subdivided icosahedra models and to test their unfoldability. I found their developments (page 7) and performed all of the testing and computation discussed in section 6, 'Empirical data'. Joe and I both agreed that his numerical estimates (section 5, 'Proof') were significantly less accurate than my testing and simulation data.

which are now guaranteed to introduce overlap. It must be emphasized that this is a lower bound, as a single assembly has 320 possible nets.

The paper then addresses two issues: to extend this result to a *mathematically provable* lower bound on the percentage of overlapping unfoldings of a polyhedrally-banded surface, and to find a *statistical* lower bound. The two results diverge, which is to be expected.

The mathematical lower bound on the percentage of overlapping unfoldings

The lower bound found through mathematical reasoning is constructed as follows[4]:

It is known that 6^2 possible nets of a locally-connected hex assembly will contain overlap. (No statement is made about the remaining 284.)

If a hex assembly is 'embedded' within a hexagonal tiling structure of 16 surrounding banded hexagons, it can be shown that the odds of the central hex assembly remaining locally-connected after cutting are at least $1/2^228$. The particular structure of the tiling, with sixteen faces per tile, was chosen to simplify the computation of this fraction. This is an *extremely* loose lower bound on the probability that the hex face will be a leaf node in the unfolded net of the hexagonal tiling structure. (For details on how this number was found, the reader is referred to O'Rourke's text in the paper.) With this loose bound, the probability p that a hex assembly will be both locally-connected and unfolded to hexagon overlap has a lower bound of $6^2/2^228 \approx 10^{-67}$.

Let $H = 20 \cdot 4^L$ be the number of hexagons in the polyhedron P constructed by recursively subdividing the twenty faces of an icosahedron L times to create a geodesic approximation to a sphere and then banding every triangle in the geodesic sphere. At most $1 - p$ of the cut-graph patterns possible for each hexagon embedded in a sixteen-hex tiling cluster avoids overlap. There are $\lfloor H/16 \rfloor$ such clusters, and so the fraction of cut-graphs that simultaneously avoid overlap in all clusters is $(1 - p)^{\lfloor H/16 \rfloor}$.

> Finally, as $L \to \infty$, $H \to \infty$, and the overlap-avoiding fraction of all unfoldings goes to 0 while the overlap fraction goes to 1. This is the main claim of the note. (BO08, p.10)

The statistical lower bound on the percentage of overlapping unfoldings

The proof above establishes a very loose bound, because overlap can occur for a number of reasons but the computations given address only a single cause. The looseness of the argument is dramatically revealed through empirical testing.

Given the number of faces involved (140 for an $L = 0$ banded geodesic sphere) an exhaustive evaluation of every possible unfolding of the banded polyhedra was infeasible. Instead, a Monte-Carlo style search was performed: millions of cut-graphs were generated at random for $L = 0$ and evaluated for overlap. Of the 5.5 million cut-graphs tested using the YAMM software (see Appendix B), only 11 cut-graphs unfolded the level-0 banded icosahedron without overlap.

[4]This portion of the material, presented on pages 7 through 9 of the paper, is primarily the work of Joseph O'Rourke, and thus is only summarized here

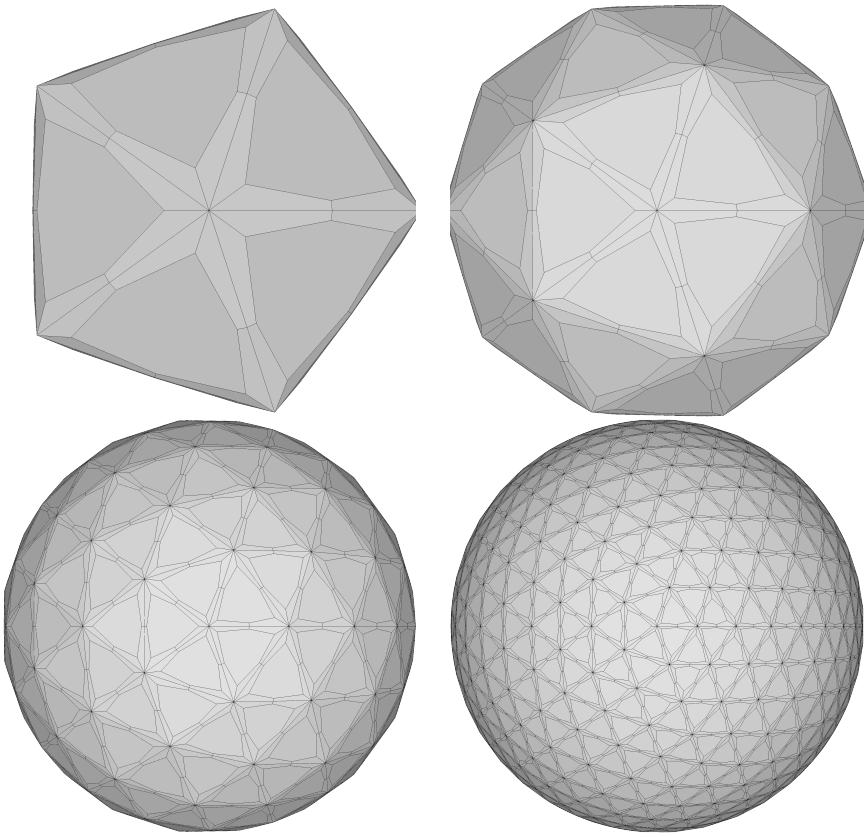


Figure 2.32: Banded geodesic spheres from (BO08), $L = 0, 1, 2, 3$.

That gives a ratio of 2×10^{-6} or 99.9998% overlap, i.e., overlap is almost never avoided; a stark contrast to the roughly 10^{-67}% rate of overlap predicted above.

This observed overlap rate of 99.9998% may be explained by the observation that, in the random unfoldings generated on the banded icosahedron, roughly 70% of the cut-graphs unfolded the seven faces of at least one hex assembly into a locally-connected net. This fraction is surprisingly stable; evaluations for higher L seemed to indicate that for increasing L the fraction would approach a value in the neighborhood of 69.75%, although this was not tested extensively. Given that the emprically-observed rate of overlap of a single hex assembly is roughly 50%, as mentioned above, one would expect roughly the fraction

$$1 - (1 - 0.7 \cdot 0.5)^H$$

of all unfoldings to overlap. For $L = 0$, $H = 20$, this formula evaluates to 99.97%, which is significantly closer to the observed fraction of 99.9998%. This also suggests that local overlap within hex assemblies is responsible for the majority of all overlaps in the banded geodesic sphere unfolding.

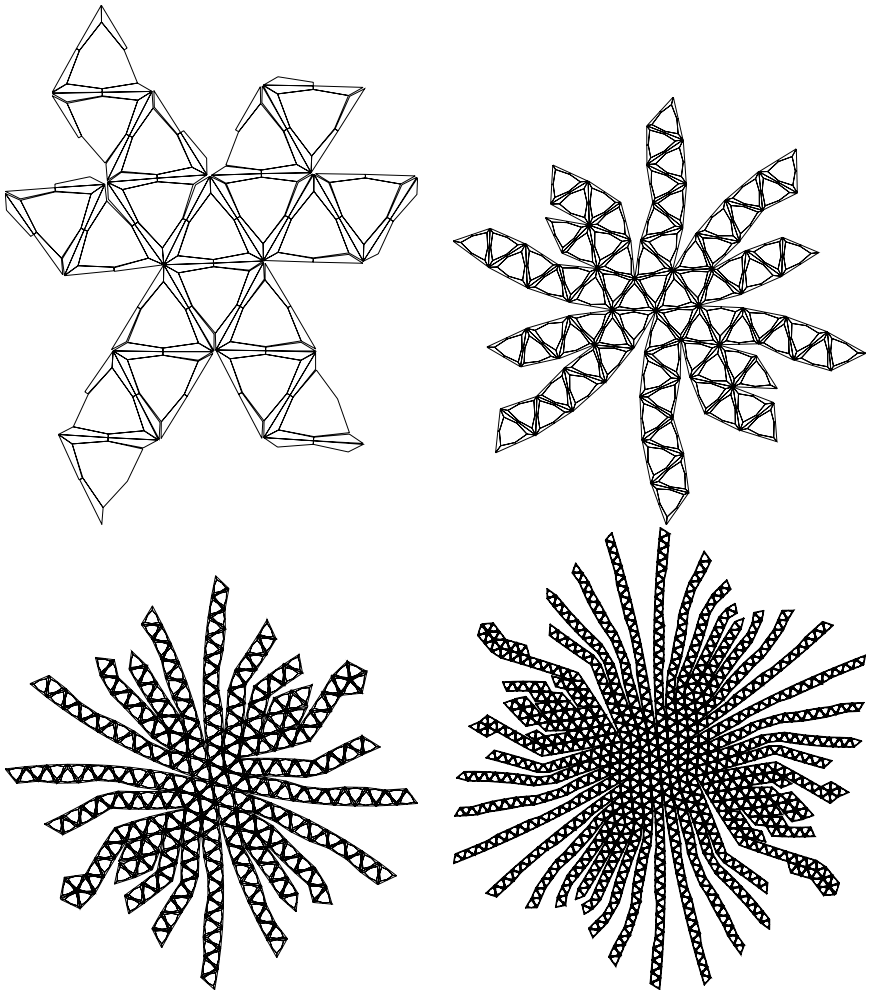


Figure 2.33: Edge-unfoldings of the first four levels of banded geodesic sphere

2.9 Conclusions

This chapter has established the following:

While cut-graphs have been demonstrated for convex polyhedra, simplicial convex polyhedra, and even 'very smooth' simplicial convex polyhedra which create overlap, it is still the case that every convex polyhedron found to date has at least one overlap-free unfolding.

Several classes of cut-graph have been defined which do *not* unfold every convex surface without overlap:

- Fukuda's conjecture that an unfolding along shortest edge-paths must be free from self-intersection has been shown to be false.
- The author's conjecture that an unfolding along an open convex curve must be free from self-intersection has been shown to be false.

In contrast, it has been shown that any *locally-convex cut-graph* will unfold a surface without overlap. However, it has been demonstrated that this class of cut-graph cannot apply to many convex polyhedra. While there is still room for exploration in this avenue, it is already quite limited.

Certain classes of polyhedra can be shown to be unfoldable through *leaf truncation*, such as the *domes*. The full class of all surfaces which may be constructed through leaf truncation has not been precisely described, although it has been shown to contain every polyhedron whose one-skeleton can be spanned by a tree which is strictly monovalent or trivalent at all but one vertex. It has been established that vertex truncation is too limited to apply to general convex polyhedra.

The Anchorable Convex Hull, an algorithmic proof for building nets without overlap, has been described. The ACH has been shown to be too limited in application for most surfaces, but it has inspired ideas which are re-visited elsewhere.

Polyhedral banding, a new method for building 'hard to unfold' convex polyhedra, has been discussed.

Chapter 3

Three Potential Proofs that All Convex Polyhedra are Unfoldable

3.1 Introduction

This chapter presents three conjectures: three potential proofs of the unfoldability of convex polyhedra. Each proof is incomplete, and in a very real sense this chapter is a 'to do' list of future research. While none of these approaches is conclusive, each represents a significant step forward and is highly deserving of ongoing investigation.

The proofs presented here come from three very different schools of thought:

- Section 3.2 explores what would be necessary for a 'classical' proof of convex unfoldability. A proof is proposed in a style similar to Euclid's proof that there is no largest prime number: to show that there is no *smallest undevelopable convex polyhedron.*
- Section 3.3 presents the *alpha-beta rules*, an algorithm and partial proof which demonstrate that it should always be possible to construct a cut-graph which can unfold a convex surface without self-intersection.
- Section 3.4 presents a partial proof that any cut-graph which unfolds a surface to overlap may be *repaired* by eliminating collisions, in a manner which is guaranteed to introduce no new collisions in regions which have already been repaired.

Each of these three proofs has great potential, but they also have weaknesses. Were these flaws to be resolved, any one of these proofs could be the demonstration that all convex polyhedra are unfoldable.

3.2 The Lower Bound of Undevelopable

One interesting approach to the proof of convex unfoldability is to model a proof in the style of Euclid's argument that there is an infinite number of prime

numbers[1]. Euclid's proof is quite simple: he shows that a conjectured counterexample cannot exist within some finite chosen range, regardless of how the range is chosen. A similar approach may be taken in the unfolding problem if a suitable range (such as the number of sides of the polyhedron) and target (such as a polyhedron for which every possible unfolding contains overlap) are chosen.

The argument proposed is as follows:

> Begin from the assumption that it is not true that all convex polyhedra are unfoldable: assume that there is some finite convex polyhedron (and therefore, in all probability, an infinite class of polyhedra) which has no self-intersection-free unfolding. If this were the case, there would necessarily be some polyhedron which had the least number of faces of all of the conjectured undevelopable polyhedra. Such a polyhedron would have at least five sides because, by Lemma 1.4, all tetrahedra are unfoldable.
>
> Let P_0 be the *smallest undevelopable polyhedron*, a polyhedron with no non-overlapping unfoldings having the least number of faces of all such undevelopable polyhedra.
>
> For any triangular face $\triangle abc$ on P_0 there must exist some polyhedron P_1 with a trivalent vertex V where if V were truncated in a certain way, the resulting face would be $\triangle abc$ and the resulting polyhedron would be P_0.
>
> P_1 is unfoldable. This is known, because P_0 has one more face than P_1 and by definition, anything with fewer faces than P_0 can be unfolded.
>
> When a trivalent vertex is truncated, the resulting triangle can be glued in the unfolding to any of its three new edges and will always fit in at least one of them. Therefore the truncation of V will not introduce conflict. Since it was known that P_1 was unfoldable, this would imply that P_0 was unfoldable, a contradiction.
>
> Therefore P_0 has no triangular face.

Similar arguments could then be made for four-sided and five-sided faces. Coupled with Lemma 1.8 (p. 18) which showed that all closed convex polyhedra have at least one face with at most five edges, this would show that the smallest undevelopable polyhedron could not exist, and thus establish that all convex polyhedra were unfoldable.

In order to execute the proof, it is first necessary to establish the condition for three-sided faces. This conjecture may be phrased formally as:

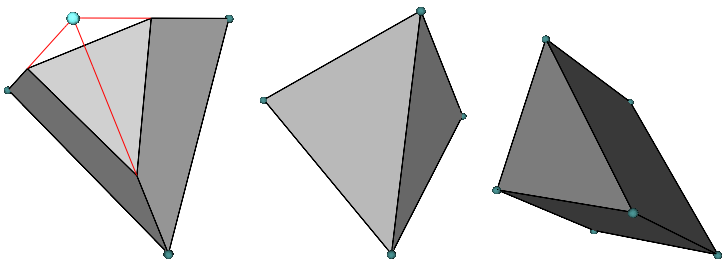
Conjecture 3.1 *The smallest undevelopable polyhedron has no triangular faces.*

Rebuttal

The proposed proof of the conjecture has two shortcomings: the less serious is

[1] Euclid's argument for the infinitude of primes, as restated by Ribenboim in (Rib95, p.3), is as follows:

> Suppose that $P_1 = 2 < P_2 = 3 < \ldots < P_r$ are all of the primes. Let $P' = P_1 P_2 \ldots P_r + 1$ and let P be a prime dividing P'; then P can not be any of P_1, P_2, ..., P_r, otherwise P would divide the difference $P' - P_1 P_2 ... P_r = 1$, which



(a) Convergent (b) Divergent (c) Parallel

Figure 3.1: A triangular face can only be achieved through the truncation of a single vertex if the edges which lead to the proposed vertex converge at a point 'beyond' the face (outside the polyhedron.)

the presumption of 'truncatability', while the more serious is the assumption that gluing will always succeed.

This argument assumes that every triangular face can be achieved by truncation. This is not necessarily the case. Consider Figure 3.1: if the edges incident to $\triangle abc$ converge at the point V outside the polyhedron then V can be truncated to yield $\triangle abc$ (Figure 3.1(a).) However in case (b) the edges converge to a point which lies 'below' the truncated face, on the wrong side of the plane that contains the face; in case (c) the projections are parallel and will never converge.

The greater flaw in the argument, however, is the assumption that after a truncation the resulting triangle can always be glued to one of the new truncation edges. This is disproved with the convex cap counterexample, shown in Figure 3.2, which was used to similar ends in (BO07), discussed on page 38. This counterexample merits further exploration.

Consider the surface P_1 shown in Figure 3.2(a). P_1 has been designed to unfold under the chosen cut-graph T (marked in red) so that it does not have the *empty sector property* (Section 2.4) as shown in Figure 3.2(d). When the highlighted vertex is truncated (Figure 3.2(b)) to produce a new triangular face (Figure 3.2(c)) without otherwise changing T, the new face f (highlighted in blue) has no place to unfold: as shown in Figure 3.2(e–g), there is no edge where the new face can be glued without overlap.

This example shows that the assumption that 'after a truncation, the resulting triangle can always be attached without overlap' is false.

This invalidates the *structure* of the proof of the conjecture, although it does not prove or disprove the original premise. The conjecture remains an open question. □

---

is impossible. So this prime P is still another prime, and P_1, P_2, ..., P_r would not be all of the primes.

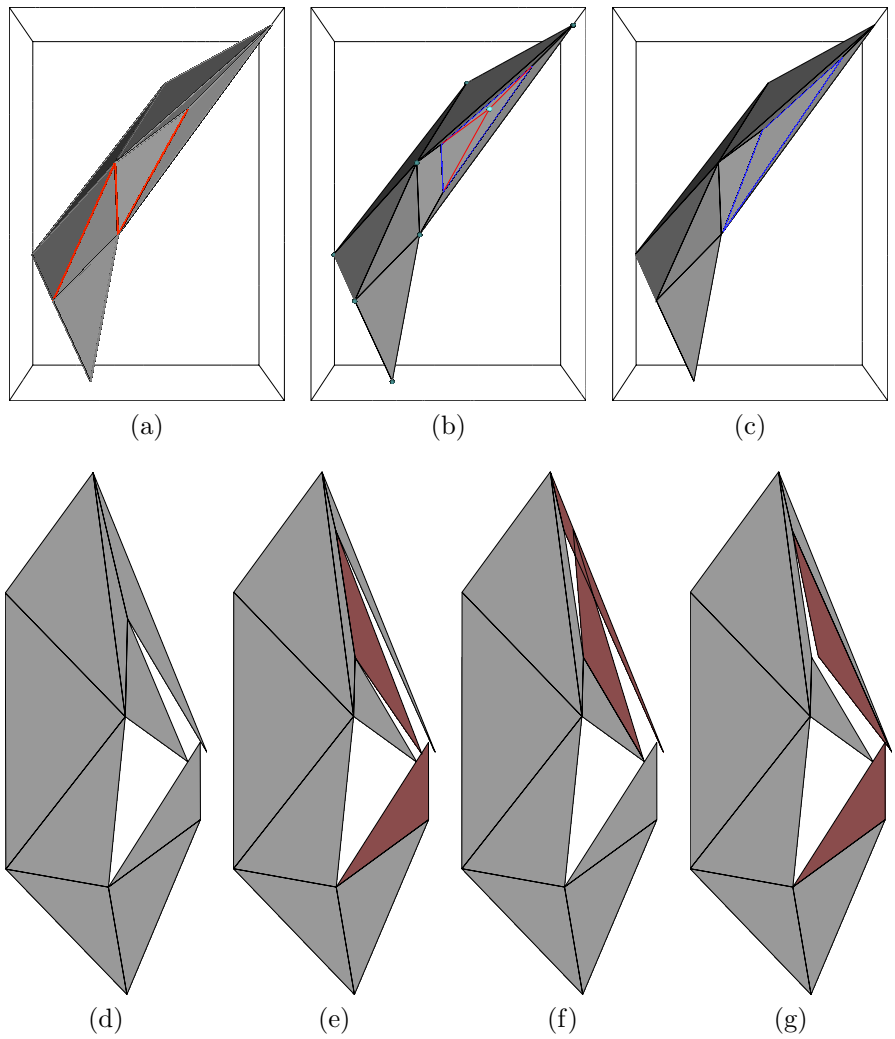


Figure 3.2: (a) The convex cap counterexample P_1, known to be unfoldable by the cut-graph shown in red (b) A single vertex of P_1 is truncated and replaced with a new face f (c) The resulting P_0 with f; f is shown limned in blue (d) The overlap-free unfolding of P_1, which does not have the empty sector property (see Section 2.4) (e–g) The overlapped unfoldings of P_0: f overlaps an adjacent face at each of its three possible anchoring edges

3.2.1 Cut-tree truncation and the smallest undevelopable polyhedron

It is interesting to conjugate the previous conjecture with the empty sector property. Consider the example given above: though the truncated vertex was a leaf-node of the graph, it did not have the empty sector property. If it had, then by Theorem 2.5 there would have been no overlap.

This shows that

Lemma 3.2 *If there exists a smallest ununfoldable polyhedron P_0 and P_0 has triangular faces, then every possible polyhedron and cut-graph (P_1, T) which can be leaf-truncated to produce P_0 must lack the empty sector property at the vertex of truncation.*

Proof. Let P_0 be the smallest ununfoldable polyhedron.

Assume that P_0 has triangular faces. Label one such face $\triangle abc$.

Let P_1^{abc} be the particular polyhedron whose vertex v must be truncated to generate P_0 with face $\triangle abc$.[2]

By assumption on P_0, P_1^{abc} is developable. The only difference between P_1^{abc} and P_0 is the removal of v and its replacement with abc, so the location in the plane of every unfolded face in P_1^{abc} will be the same in the unfolding of P_0. This means that any conflict (and its is known that there is a conflict) must come from $\triangle abc$.

Furthermore, this is a conflict which will appear *for every possible valid cut-graph of* P_1^{abc}, because if there existed a single cut-graph which could unfold P_1^{abc} in such a way that the truncation of $v \rightarrow \triangle abc$ produced no overlap then P_0 would be unfoldable.

Thus if P_0 is the smallest ununfoldable polyhedron and P_0 has triangular face $\triangle abc$ achieved by truncating the vertex v from the smaller polyhedron P_1^{abc}, then the vertex v does not have the empty sector property under *every possible cut-graph which unfolds* P_1^{abc}. □

It is somewhat difficult to categorize this result. It does not eliminate the possibility that there is an undevelopable polyhedron with triangular faces, and in this sense it is quite weak; and yet, it places an almost absurdly stringent requirement on the conjectured surface, one which would seem intuitively to be almost impossible to meet, and in this sense the lemma is surprisingly strong.

3.3 Angular Restrictions to Ensure Developability

In (Luc06, p.54), Brendan Lucier illustrates the angular requirements for *2-local overlap*, the simplest possible form of self-intersection in the unfolding of a convex polyhedron. Lucier shows that if a branch of the cut-graph 'curls' too much upon itself then overlap becomes possible. Lucier's goal was to demonstrate how to create overlap, but it becomes interesting to turn the argument around

[2]Note the assumption that there exists exactly one such P_1^{xxx} for each triangular face $\triangle xxx$ in P_0. This is not an accurate assumption, for the reasons detailed above in the first objection to the conjecture (that is to say, P_1^{xxx} may not exist at all) but this inaccuracy is immaterial to the substance of the discussion, and will be disregarded.

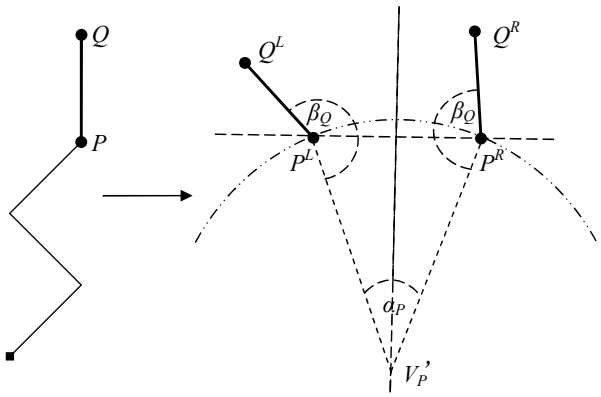


Figure 3.3: Unfolding the branch Φ. The angle between P^LQ^L and P^RQ^R is α_Q. The lesser of the angles from V_P' to each of the developments of PQ is β_Q.

and ask: is there an upper bound on how sharply a cut-graph branch can turn, below which there is no risk of self-intersection?

This section presents an extension of the so-called 'Alpha-Beta Rules', a set of lemmas originally presented to the Sixth International Conference on Curves and Surfaces, Avignon 2006 as *Angular Restrictions to Ensure Developability* (Ben06). The original work was, unfortunately, done before the author became aware of Lucier's research, and in their original form the alpha-beta rules were strictly limited by the particular geometry of the surface to unfold. The material presented in this section represents a step beyond those restrictions.

At Avignon, the alpha-beta rules were described as being, "designed to show that it is possible to 'grow' a cut-graph from the outermost leaves inwards to form a complete, spanning tree guaranteed by its incremental assembly to unfold without overlap.". Strictly speaking, this was a misleading summary: there is no 'time' in cut-graphs, thus no growth. A more accurate description of the intent of the alpha-beta rules would have been to say that they are designed to provide a *partial ordering of the vertices of the polyhedron*, established by a set of sequential choices, such that any unfolding which follows this ordering will be free of overlap.

This section presents the building blocks of a proof. The proof is not complete, and the work which remains to be done is detailed before the end of the section.

3.3.1 Terminology

Given the nature of the task, much of the following discussion will revolve around some partially-formed cut-graph and the task of selecting which vertex to add to it next. It will be assumed that the leaves of the cut-graph are 'fixed' and that each new node to discover will be a step away from the leaves of the graph and toward the root; the reader might picture a set of streams, emerging from sources around the slopes of a valley and gradually merging together into larger rivers. Thus for any subset of the cut-graph, there will be a single vertex which marks the current 'river mouth', with the path downstream as yet unknown.

As the phrase 'subset of the cut-graph' is cumbersome to wield, the shorter term *cut-path* will be used here to describe any connected subset of some cut-graph T which is to be discovered on a convex polyhedron. In the original publication of (Ben06) a cut-path was limited to being a connected and Hamiltonian path, but that restriction is lifted here. A cut-path may contain multiple leaf nodes of T. All leaf nodes of a cut-path are treated as fixed, and each cut-path has a single vertex which is the *tip* of the cut-graph. The *tip* is the node the furthest from the leaf nodes; all growth of a cut-path, and union with other cut-paths, occurs at the tip. In this section, the cut-path currently being discussed will be labeled 'Φ' and its tip will be labeled 'P' (Figure 3.3.)

Let α_P be the sum of the angle deficits of the vertices of Φ, up to *but not including* the angle deficit of P. Recall that the angle between the two developments of any edge is exactly the sum of the angle deficits of all cuts which lead to that edge, and that the total angle deficit of any surface of genus zero is 4π. α_P is not defined if P is a leaf node of Φ (that is, if Φ consists of a single vertex.)

Let the *transverse line* of P be defined as the line which passes through the points P^L and P^R.

The goal will be to find a new vertex Q to add to Φ beyond P, 'downstream' of P. The intent is that the developments of edge PQ should not cross each other, nor cross the developments of any other edge upstream in Φ.

Let β_Q be the turn angle between the cut-path and the developments of PQ. To speak of the 'turn angle' from what could easily be a tree with many leaves is quite imprecise, and so a new concept is introduced to refine the definition of β: the *virtual root*.

Note: In each of the following lemmas, it is assumed that α_P is less than π. The case where $\pi \leq \alpha \leq 4\pi$ is discussed below.

3.3.2 The virtual root

When unfolded, P will develop to two distinct points in the plane, P^L and P^R. These points define a *virtual root*, V'_P, an idea first encountered as 'collapsing a fork' in the discussion of locally-convex cut graphs in Section 2.6 . V'_P is defined to be the point[3] in the plane which lies on the perpendicular bisector of $P^L P^R$ to the right of the ray $P^L \rightarrow P^R$ at a distance d from P^L and P^R chosen such that the angle between the line segments $V'_P P^L$ and $V'_P P^R$ is exactly α_P.

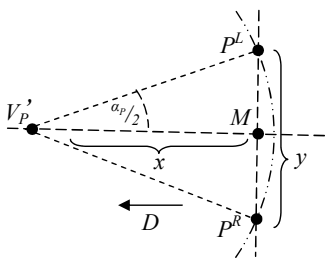
Given V'_P, for any vertex Q which is being considered for addition to Φ beyond P, β_Q is defined to be the lesser of the two angles $\beta^L_Q = \angle V'_P P^L Q^L$ (measured counterclockwise) and $\beta^R_Q = \angle V'_P P^R Q^R$ (measured clockwise.) Note that β_Q is always positive.

In many developments of an edge, one edge will 'turn away' further than the other. Formally, if β^L_Q is less than β^R_Q then the *inner development* of PQ

---

[3] Calculating V'_P:

$$
\begin{array}{lcl}
M & = & (P^L + P^R)/2 \\
y & = & \|P^L - P^R\|/2 \\
D & = & ((P^L - P^R) \times [0,1,0])/y \\
x & = & (y/2)\cot(\alpha_P/2) \\
 & & \\
V'_P & = & M + xD
\end{array}
$$



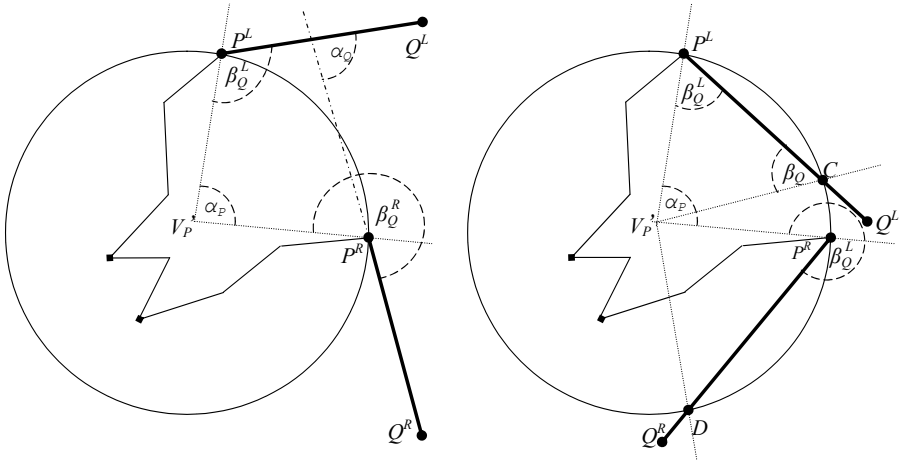


Figure 3.4: If $\beta_Q > \frac{\pi-\alpha_P}{2}$ then there can be no conflict between the left and right developments of PQ. (a) $\beta_Q \geq \frac{\pi}{2}$ (b) $\frac{\pi-\alpha_P}{2} < \beta_Q < \frac{\pi}{2}$

is P^LQ^L and the *outer development* of PQ is P^RQ^R; and vice-versa, if the comparison of the angles is reversed. The inner development of PQ may be denoted by P^IQ^I.

Let Γ be the circle of radius d centered on V_P', passing through P^L and P^R.

3.3.3 Preventing overlap within each pair of unfolded edges

Lemma 3.3 (Alpha-Beta Rule 1) *Given a cut-path Φ with tip vertex P, virtual root V_P', and a potential extension vertex Q to be considered for addition to the cut-path beyond P, if $\beta_Q > \frac{\pi-\alpha_P}{2}$ then it is impossible for P^LQ^L and P^RQ^R to intersect.*

Proof.

Lemmas One and Four of (Ben06, p.3) gave a detailed mathematical proof that if $\beta_Q \geq \frac{\pi-\alpha_P}{2}$ then P^LQ^L and P^RQ^R cannot cross; the arguments presented in the paper hold just as well for V_P' as for V. The proof given was difficult to interpret and so a more geometric argument is offered here.

Case (a) — $\beta_Q \geq \frac{\pi}{2}$:

Consider Figure 3.4(a). If $\alpha_P = 0$ then the two developments would be coincident, but on a convex surface $\alpha_P > 0$. As α_P increases, the distance from P^L to P^R must increase by the length of the chord of arclength α_P on Γ; simultaneously, Cauchy's Arm Lemma (p. 18) dictates that as the angle α_P grows, the distance from Q^R to Q^L must increase. Thus (up to $\alpha_P < \pi$) the two developments can never cross.

Case (b) — $\frac{\pi-\alpha_P}{2} < \beta_Q < \frac{\pi}{2}$:

If $\beta_Q < \frac{\pi}{2}$ then the edge P^LQ^L intersects Γ at some point C (Figure 3.4(b).) The triangle $\triangle P^LCV_P'$ is an isosceles triangle with angles β_Q, β_Q, $\pi - 2\beta_Q$.

The angle of the triangle at V_P', between the lines $V_P'P^L$ and $V_P'P^R$, is α_P. The angle deficit of P itself does not factor into the angle between the

developments leading up to P.

If $\alpha_P > \pi - 2\beta_Q$ then P^R must lie outside the triangle $\triangle P^L C V'_P$, in which case the triangle $\triangle P^R D V'_P$ does not overlap $\triangle P^L C V'_P$ at any point other than V'_P.

Therefore if $\beta_Q > \frac{\pi - \alpha_P}{2}$ then $P^L Q^L$ and $P^R Q^R$ cannot intersect. □

Corollary 3.4 *Given a cut-path Φ with tip P and potential extension vertex Q, the angular restrictions to avoid overlap between the developments of PQ apply only to the inner development, which has the lesser turn angle in the plane. The remaining development may turn freely: $AD(P)$ is irrelevant to the potential overlap of the developments of the edge PQ.*

Proof. β_Q is defined to be the minimum of the turn angles of the left and right developments, measured counterclockwise and clockwise respectively; label these two angles as β_Q^L and β_Q^R. Lemma 3.3 guarantees that if $min(\beta_Q^L, \beta_Q^R) > (\pi - \alpha)/2$ then there is no risk of overlap between the developments of this edge, leaving $max(\beta_Q^L, \beta_Q^R)$–which determines the position of the outer development– unconstrained. □

Corollary 3.5 *Given a cut-path Φ with tip P and potential extension vertex Q, if $\beta_Q > \frac{\pi - \alpha_P}{2}$ then the developments of PQ will not cross the 'upstream' developments incident to P^L and P^R.*

Proof. Recall that it has been assumed that $\alpha_P < \pi$, placing V'_P to the right of the ray $P^L \rightarrow P^R$. By the same token, the two edge-developments leading to P^L and P^R must also lie to the right of $P^L \rightarrow P^R$. The proper choice of Q must either place the developments of PQ to the left of $P^L \rightarrow P^R$, or to the right of $P^L \rightarrow P^R$ but outside the line segment $P^L P^R$. Thus there can be no intersection between the developments of PQ and the other two developments incident to P^L and P^R. □

3.3.4 Extending a cut-path

Having shown that there are broad angular constraints within which an edge can be cut without risk of overlap, it can now be shown that an edge which meets these constraints must always exist:

Lemma 3.6 *Given a cut-path Φ ending in tip vertex P, there must exist at least one vertex Q such that $\beta_Q > \frac{\pi - \alpha_P}{2}$ for the edge PQ.*

Proof.

Consider Figure 3.3, in which the transverse line through $P^L P^R$ divides the plane in two; call the side containing V'_P the 'upstream' side, opposite the 'downstream' side. Recall that the angle between any two developed edges which share a common developed vertex is the sum of the face angles between them.

Then if there were no edges in the downstream half of the plane (Figure 3.5) it would imply that two edges met at P^L or P^R with no edge of the polyhedron between them but also with more than π radians of incident face angle. This would mean that the face shared by the two edges had $> \pi$ incident angle at a vertex, which is impossible on a convex polyhedron. No face of a convex polyhedron may itself fail to be convex.

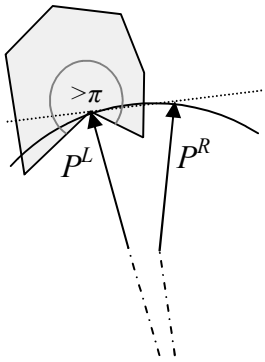


Figure 3.5: There must exist at least one vertex with $\pi/2 \leq \beta < 3\pi/2$; for this not to be the case would imply that a face incident to P was nonconvex.

If there is an edge in the downstream half of the plane, β for that edge would be between $\pi/2$ and $3\pi/2$. Even if α were zero, this would still be sufficient to guarantee that there is no overlap in cutting that edge. □

3.3.5 Preventing overlap between an edge and previous edges on the cut-path

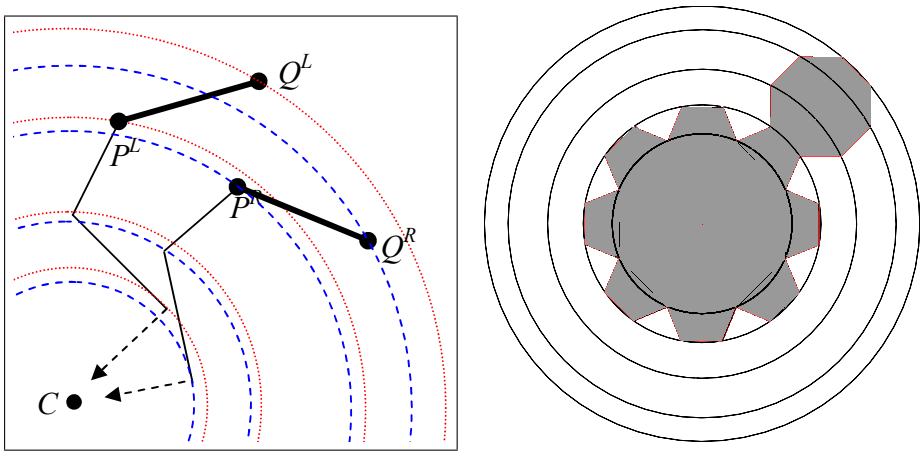
Lemma Two of (Ben06, p.4) proved that, under certain stringent conditions, each new edge added to a cut-path would not cross *any* development of an edge already in the cut-path. Unfortunately, those conditions were too strict: the lemma required that (a) Φ was Hamiltonian and thus had exactly one root, labelled V; and (b) that that root was coincident with the virtual root V'_P.

Thus far, Lemmas 3.3 and 3.6 and the associated corollaries have shown that if Q is chosen to have $\beta_Q > \frac{\pi - \alpha_P}{2}$, then PQ will cross neither itself nor its immediate predecessor in the cut-path, and that there must always exist at least one Q which fits the bill.

However, the restriction on β is not sufficient to guarantee that there will be no overlap with prior edges in the cut-path. This is because the restriction on β is strictly local; to avoid overlap there must also be a global factor guiding the choice of edge. Inspired by the design of the Anchorable Convex Hull, what is needed is some central anchoring point or face.

The *center of the unfolding*, C, is an arbitrarily-chosen point, somewhere within the border of the net, which will determine a focal direction for the unfolding. The center of the unfolding may be chosen from the development of any point on the polyhedral surface. In the software implementation of the alpha-beta rules in YAMM, the author has chosen to use the center of the development of the lowest face in the polyhedron, but any point which falls within the first developed polygon will do.

Lemma 3.7 (Alpha-Beta Rule 2) *Let Φ be a cut-path with tip P, where every edge already in Φ has been chosen according to the constraint presented in this lemma. Let Q be a potential extension vertex to Φ, and label the inner development of PQ (be it the left or right development) as $P^I Q^I$. Let C be the*



(a) (b)

Figure 3.6: (a) Each development from $P \rightarrow Q$ is bounded by concentric rings around V (left-hand developments are shown on red dotted lines, right-hand on blue dashed lines) (b) Concentric rings around the center of the radially-oriented unfolding of the shallow truncated octahedral pyramid

center of the unfolding, a point within the development of the root face of the unfolding. Then, if Q is chosen such that

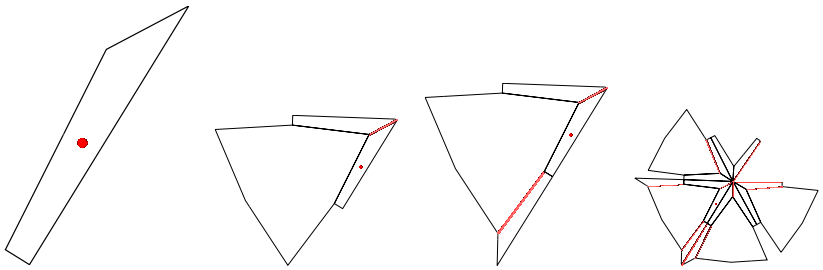
- $\beta_Q > \frac{\pi - \alpha_P}{2}$
- *Q maximizes the value of the dot product* $\frac{CP^I}{\|CP^I\|} \cdot \frac{P^IQ^I}{\|P^IQ^I\|}$ *above any other vertex in the 1-ring of P*
- *The dot product is greater than zero*

then the left-hand development of PQ will not cross any left-development of any edge already in Φ, and the right-hand developments likewise will not cross.

Proof. Each new vertex added to Φ will be chosen such that the vector from the inner development of P to the associated development of Q is a vector which points, as nearly as possible, directly away from the center. It has already been established that at least one Q must exist; this is a criterion for distinguishing between multiple valid options, should they be available.

The inner development of each new vertex added to Φ will always fall at a greater distance from C than its predecessor if $\frac{CP^I}{\|CP^I\|} \cdot \frac{P^IQ^I}{\|P^IQ^I\|} > 0$ (Figure 3.6(a).) If each new vertex develops to progressively further radii from C then they can never cross a preceding edge. □

By the same logic as that used in Lemma 3.6, there must always be at least one Q which would travel away from C, although it is less clear that such a Q would also comply with the rules on β.



(a) Step zero (b) Step one (c) Step two (d) Step eight

Figure 3.7: Four steps in the discovery of a cut-path on a banded icosahedron. The red dot marks the root face.

3.3.6 Cut-paths and partial unfoldings

Before proceeding, further discussion of what it means to 'cut an edge' is warranted. In the scenarios discussed here, the developed edges of a cut are often treated as simple joined line segments in the plane; but the reader must recall that they are in reality the outer edge of a polygon which has been unfolded in its entirety. To cut an edge is to partially determine the relative positions of the faces nearby, which can be exploited when developing a surface. This insight has interesting implications.

For example, consider a leaf-node of the cut-graph. Because only one edge incident to the leaf-node is cut, the fan of faces around the vertex are locked together; their relative placement is fixed and immutable.

Likewise, consider a vertex cut by two incident edges. The two arcs of faces separated by the cuts are still bound together within themselves; the positions of all the faces to the left of the cut are fixed relative to each other, and likewise for the right.

Recall from p. 11 that a *partial unfolding* U is a connected subset of the unfolded faces of a polyhedron. Each cut-path Φ defines a unique partial unfolding U_Φ: U_Φ is the unfolded net of faces which are incident to a vertex in Φ. The edges of Φ are a subset of the perimeter edges of U_Φ.

To find the partial unfolding U_Φ determined by a cut-path Φ, the partial unfolding is constructed iteratively. Begin with any face f_0 incident to any vertex V_0 which is a leaf-node of Φ. Develop f_0 into the plane; this will be the root face of U_Φ. Thereafter, for every face f which is developed, for each undeveloped face g which is adjacent to f across a shared edge $e \notin \Phi$ where g is incident to a vertex of f which is also a node of Φ, develop g by gluing its image to the development of e.

This process is illustrated in Figure 3.7, which shows several early steps in the discovery of cut-paths to unfold a banded icosahedron. Faces are added as each new vertex is expanded; in time, the separate cut-paths will merge together to form a single complete cut-graph.

3.3.7 Vertex ordering and interdependency

The lemmas presented thus far have focused on the extension of a single cut-path from a single tip. However, such focus may be misleading, because vertices are not extended in isolation. Consider: to choose the best outbound edge to cut from a given vertex P, α and β must be calculated. To find α requires prior knowledge of every edge which will be cut up to P in the cut-path. Thus the expansion of P is dependent on the prior expansion of every other vertex in the cut-path before P.

For example, what would it mean to cut an edge from P to Q if Q has already been expanded? Doing so changes the value of α_Q at Q by adding new incident cut edges. This in turn could mean that some new edge which had previously not met the constraints on β_Q is now a candidate for expansion, perhaps pointing more directly away from the center. Then Q's expansion must be recomputed, which could mean re-expanding every vertex past Q in the cut-path.

The solution is two-fold: *sort vertices* by increasing distance from the chosen center point, and support a limited amount of *backtracking*.

Sorting vertices for expansion

The goal in sorting vertices by distance is to minimize–or, if possible, eliminate–the number of occasions where a vertex is expanded by cutting to a vertex which has already been expanded. As every expansion will point away from the center according to Lemma 3.3, it makes sense to first choose vertices nearest to the center. Assume that the unfolding has begun at the lowest face of the polyhedron F_0 and that the center point C of the unfolding has been chosen to be the center of the development of F_0. There are then several sorting options available:

1. Sort vertices lexicographically with height as the primary axis.
2. Sort vertices by geodesic distance from the center of F_0, breaking ties with lexicographic order.
3. Sort vertices by chord-length inside the polyhedron (linear distance in space)
4. Sort vertices by the minimum of the radii of the developments of each vertex
5. Sort vertices by the maximum of the radii of the developments of each vertex

After fairly extensive experimental trial and error, the author has chosen option (5) for the YAMM software, scoring each vertex by the maximum of the radii of all of its developments and then sorting all vertices by increasing score, developing the lowest-scored vertex first. It was found that this method yielded the lowest error rate.

To implement this sorting method, the vertices of the polyhedron cannot be scored ahead of time: the radii to which a given vertex develops depend completely on the developments of prior edges on the cut-path. However, this is acceptable: with each step of the algorithm, only the nearest unexpanded vertex needs to be identified, and there will always be unexpanded vertices on the outside border of the partially-unfolded net until the net is complete.

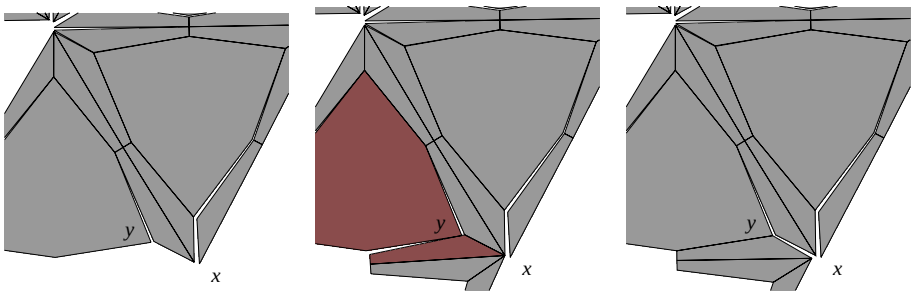


Figure 3.8: Three sequential frames from the unfolding of a banded icosahedron by the alpha-beta rules. In the course of expanding vertex x from frame (a) to frame (b), several faces are unfolded to the plane which break the β rule at vertex y. y has not yet been developed; if it had, the edge xy would already have been cut, and the expansion of x could not have restored it. From frame (b) to frame (c), y is expanded and the error is automatically rectified: the edge xy is now part of the cut-graph.

Limited backtracking

Experiment has verified that although sorting method (5) minimizes the average incidence of cuts ending at already-expanded vertices more effectively than any of the other options, it does not eliminate it. This is demonstrated in Figure 3.8, where a partial unfolding grows into–and then out of–self-intersection while following the alpha-beta rules.

In Figure 3.8(a) the vertex x is about to be expanded. The best outbound edge is found, cutting downwards and to the left in the figure (the center, not shown, is upwards and to the right.) In Figure 3.8(b) the expansion of x has added three more faces to the net, creating a less-than-$\pi/2$ turn in the cut-path around adjacent vertex y which has not yet been expanded. In Figure 3.8(c) the ordering of expansion has reached y and the best (in fact, only) edge which can be cut from y is the edge xy; β at y returns to a valid value. The sole concern is that in expanding y after x, y's new cut has altered the total α feeding into x. x should now be re-expanded, although in this case (and in the majority of cases, during testing) this re-expansion will not change the choice of outbound edge.

This demonstrates that there is an *interdependence* between some sets of vertices: situations where one vertex must be expanded before another, even though the distance-based ordering would address them in the wrong order.

This dependence relation could presumably be expressed as a dependency graph between vertices. However, this graph would change from step to step as cut-paths were added or merged; it seems improbable that a polyhedron-wide 'dependency map' could be built for a surface without constructing the unfolding beforehand. One version of the YAMM software implementation of the alpha-beta rules explored this possibility. At every step the algorithm would simulate every possible expansion of each of the vertices on the rim of the partial net and then choose to expand only from amongst vertices whose 'best' expansions would

not develop faces across edges which another vertex would later need for its own 'best' expansion. In testing, this subroutine proved to be cumbersome, difficult to extend, and insufficient to the task; it only provided 'lookahead' for one edge, where even the banded icosahedron has vertices which are interdependent across two or even three adjacent edges.

Rather than take the software down the fruitless path of unlimited-depth predictive search, a simpler solution was instead chosen, which has yielded the frames shown in Figure 3.8. For every vertex, always cut the 'best' edge according to the rules for β and C, even if that edge was previously laid down unbroken by another vertex's expansion; but if such an edge must be cut, then backtrack one step and re-expand the impacted second vertex.

Although it will not be formally proved, it seems reasonable to argue that this backtrack operation must terminate locally without inducing an uncontrolled backtrack cascade. This is because each vertex which is re-opened for backtracking will still cut an edge to point outwards, away from C. The newly-cut edge should not touch the vertex which caused the backtrack.

3.3.8 When α is greater than π

Throughout this discussion of the new alpha-beta rules, three quarters of the problem have actually been ignored: the rules above are for when α is less than π, but the total angle deficit on a closed convex polyhedron is 4π. What of the rules for $\pi \leq \alpha \leq 4\pi$?

It is tempting, in all honesty, to dismiss the question. Consider: unfolding will begin at the lowest face, with a cut-path extending from each vertex, so there are already at least three cut-paths from the very first step. If they stay separated until the top of the surface these cut-paths will each account for (on average) $4\pi/3$ radians' worth of total angle deficit; one more cut-path independently reaching the top and that average drops to π and the rules above apply.

Another reason to disregard $\alpha > \pi$ is that overlaps occur so rarely for higher values of α. The vast, vast majority of overlaps are local events, n-local collisions for small n; these collisions are overlaps within cut-paths which are small parts of much larger unfoldings. By the time α has grown to π the two sides of the development are across the unfolding from each other; clashes between them could not happen without crossing all of the intervening faces first.

And yet, neither of these objections is sufficient support for arguing that higher values of α can be disregarded entirely–not if the goal is a robust proof. To complete the proof, the rules given above must be extended to greater α, but this will not be attempted in this dissertation.

3.3.9 Weaknesses in the argument

The arguments which have been presented here are not sufficient to constitute a formal proof that the alpha-beta rules can unfold every convex polyhedron. The author deeply regrets that the proof remains incomplete.

The known flaws in the argument are:

- The reasoning that there must exist an outbound edge which satisfies both the rules for β and for C is tenuous: it holds up well for Hamiltonian cut-paths but it is, in the author's opinion, not yet strong enough for the

case where two or more edges of a cut-path flow into a single vertex to be expanded. Consider the volcano unfolding of a cone: the transverse line through the developments of the tip are poorly defined, but worse, the transverse line between any two non-adjacent developments is crossed by the development of the intervening face.

- Malcolm Sabin has provided an argument[4] that without Lemma 3.7, Corollary 3.5 must also require that the outer development of PQ fall to the right of the transverse line $P^L \rightarrow P^R$.

- Lemma 3.7 is actually very limited: it guarantees that there will be no overlap between developments *on the same side* but not between left and right sides. Without this, the proof is incomplete. The argument seems feasible: there is a relationship between the ordering of the left-hand and right-hand (red and blue in Figure 3.6) concentric circles which bound the developments of each new edge. It may be possible to exploit this relationship to use Lemma 3.3 and Corollary 3.5, perhaps by showing that there will never be more than two sequential red rings before the next blue (or vice versa).

- The reasoning that a vertex Q can be added to extend a cut-graph without overlap does not show that a second edge can be cut from some other cut-path to Q without causing intersection between the two cut-paths.

- As described above, the ordering of vertices is still a topic of open research. It would be nice to be able to justify the choice of the particular ordering chosen with a better reason than 'experimental testing'.

- A significant improvement to the backtracking support would be to undo any face developments as part of a backtrack, in addition to flagging downstream vertices for re-appraisal. The difference would be that those downstream vertices had already developed their faces and those faces then influenced the developments of other nearby vertices; all of those vertices should be rolled back too.

- Formal support for $\alpha \geq \pi$ is essential.

- Despite the author's assurance that backtrack could not induce an infinite loop, this is not actually quite true: on very rare occasions in testing, the author found that the *topmost* face of the surface would circle around the outer rim of the unfolding, looping forever as one of its vertices was developed and the other two marked for re-development continually. This is a software glitch, of course, but it does highlight the need for better-defined termination guides in the rules.

- There are still a few very rare and hard-to-categorize cases where the software will fail to find an overlap-free unfolding, despite following the rules above. These cases seem to always be related to the poorly-defined termination conditions and issues related to backtracking.

[4] Personal communications, 2008

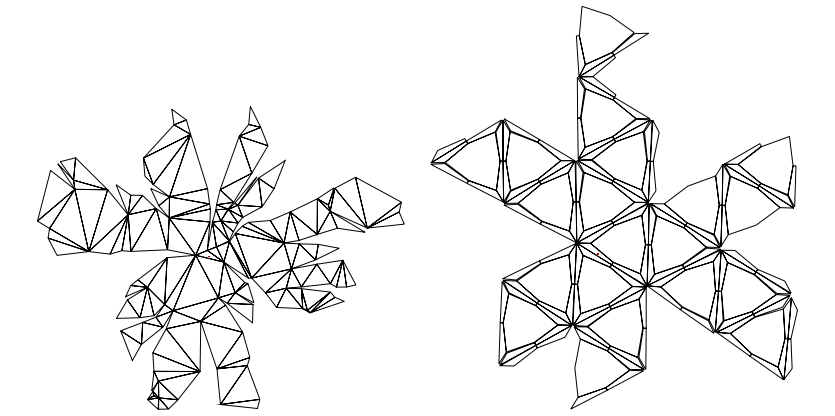
3.3.10 Experimental support

The alpha-beta rules have been implemented in the YAMM testbed with remarkable, but not perfect, success. The algorithm can unfold extremely 'difficult' surfaces such as the banded third-level subdivision of an icosahedron; which is notable because it does so *entirely without testing for collisions.* This really is a striking result, all the more so because overlap *is* generated in the course of the unfolding–but it is repaired as the process continues. In short, the nets are clearly being generated according to rules which do prevent overlap.

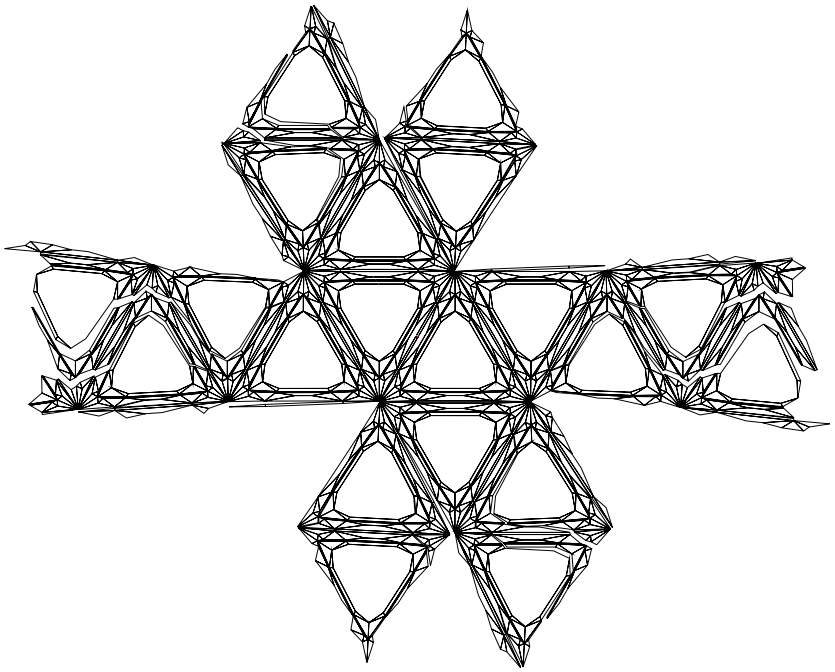
However, there are still difficulties in the software with the final steps of the unfolding. As the end of work draws near on this dissertation, the final YAMM implementation of the alpha-beta rules has a success rate of 202,376 successful unfoldings out of 203,556 randomly-generated simplicial convex polyhedra, roughly uniformly distributed from 4 to 300 faces. That gives a success rate of 99.91%: impressive, but not ideal. Analysis of the failure cases suggests that the flaw lies in the interdependency of vertices in the penultimate stage of unfolding.

Algorithm A.20, a formal algorithmic declaration of the implementation of the alpha-beta rules, can be found in Appendix A on page 175.

Figure 3.9 shows three successful unfoldings.



(a) (b)



(c)

Figure 3.9: (a) A randomly-generated convex polyhedron, unfolded by the alpha-beta rules. (b) The alpha-beta unfolding of the banded icosahedron. (c) The alpha-beta unfolding of the *doubly-banded* icosahedron, a banded icosahedron whose faces have, in turn, been polyhedrally banded. Note the cracks spreading as directly as possible away from the center of the unfolding.

3.4 Collision Repair

> *Sketch of the proof*: It is shown that any one collision in an overlapping unfolding can be *repaired* by moving the collided faces, although this may introduce new collisions elsewhere. It is then argued that on certain types of nets, new errors are introduced no nearer to the center (and often further out) than old ones, ensuring that if errors are repaired beginning with those closest to the center that they can ultimately all always be repaired.

This section presents the building blocks of a proof. The proof is not complete, and the work which remains to be done is detailed before the end of the section.

3.4.1 Terminology

Flow on a cut-graph In the discussion of this section it will be useful to describe the ordering of vertices on a cut-graph. Returning to the common metaphor of the cut-graph as a set of streams emerging from a mountainside and flowing downhill away from its leaf-nodes, the terms '*upstream*' and '*downstream*' will be used to describe the relative positions of two vertices. Specifically, for two vertices P and Q in the cut-graph T, if the longest path on T from any leaf-node of T to P is longer than the longest such path to Q then P is said to be *upstream* of Q. To extend the metaphor, this means that if one were to sail a boat from outside an unfolded net into the gap between the outermost developments of a branch of the cut-graph, one would pass the developments of Q before reaching those of P.

k-local overlap In (Luc06, p.34), Lucier defines *k-local overlap* as follows:

> Suppose P is a polyhedron with an unfolding [net] U. Suppose further that there is an overlap between faces f_1 and f_2. Then if there are at most k vertices in the shortest path along edges of U starting with a vertex incident to f_1 and ending with a vertex incident with f_2.[...] In particular, an overlap is *1-local* if f_1 and f_2 are both incident with a common vertex.

Using this definition and reasoning akin to that of Polthier in (Pol03) (see Lemma 1.2, p. 15), Lucier then gives the following useful lemma:

Lemma 3.8 (Lucier) *No unfolding of a convex polyhedron contains a 1-local overlap.*

The dual of a developed edge Every cut edge unfolds to a right development and a left development. The *dual* of the left development is the right development, and vice-versa.

The parent of a face Every unfolded net is a connected, acyclic graph: an undirected tree. Define the *parent* of a face P in an unfolding as the face which is adjacent to P in the unfolding and which lies toward the root of the net.

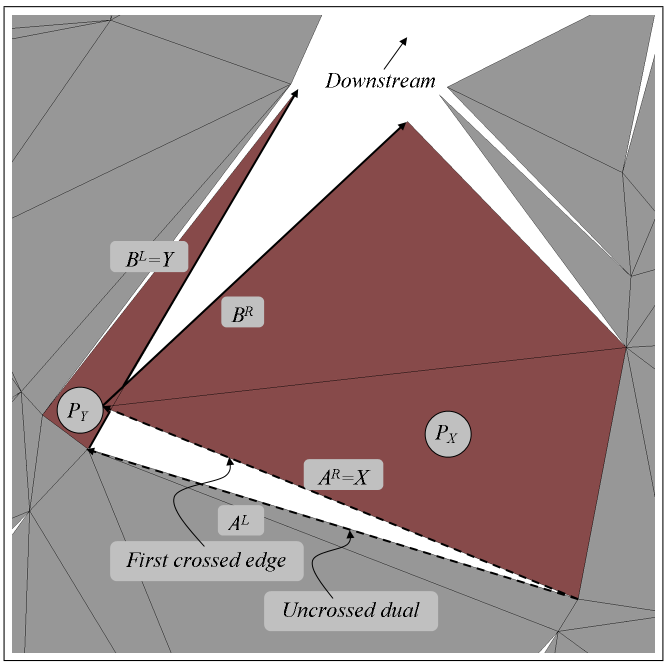


Figure 3.10: Detail of a collision showing labeled terms. There is exactly one *first crossed edge* in every collision.

If P lies on the path in the net from Q to the root of the net then P is an *ancestor* of Q.

3.4.2 The *first crossed edge*

Oftentimes when a net unfolds to collisions, more than two polygons are caught in the same group of overlapped faces (see Figure 3.11, in which each collision involves three faces.) For every such group of collisions there is always an edge which must be crossed before all others as the cut flows downstream. The *first crossed edge* is defined to be the edge which lies furthest upstream of those which are crossed in the collision, whose dual is not crossed, and whose dual lies upstream from the collision (Figure 3.10.)

Lemma 3.9 *For every group of overlapped faces, there must exist exactly one first crossed edge.*

Proof. By Lemma 3.8, no unfolding of a convex polyhedron contains a 1-local overlap. Every overlap is therefore at least 2-local and so there must exist at least one edge on the cut-graph which is not crossed and which is separated from the outside of the unfolding by the collision. Borrowing once again from the maritime metaphor, call this edge and the set of all adjacent edges in the border of the unfolding, up to but not including the first edges caught in collision on either side of the cut-graph branch, the '*lagoon*' of the collision. Lucier's corollary guarantees that in any overlap, the lagoon is not empty.

At the mouth of the lagoon are two edges which are crossed, either by each other or by other edges in the collision. Label these two edges A and B. Assume without loss of generality that A lies upstream from B (Figure 3.10.) Then A^R lies upstream from B^R on the right development of the branch of T which contains A and B, and A^L lies upstream from B^L on the left. If it were the case that A^L were also conflicted then the conflict of A^L would be further upstream (into the lagoon) than the conflict of B^L, in which case B^L would not have been the first collided edge on the left-hand shore–a contradiction.

Thus, for every lagoon there is exactly one *first crossed edge*.

The one exception would be if A were the same edge as B. However, this is impossible. The left and right developments of every edge diverge in the unfolded plane by the sum of the angle deficits of all of the vertices on the cut-graph branch upstream from them, which must be a positive angle on a convex surface. Then the two edges can only cross if they do so at a positive angle (measured counterclockwise from the direction of travel of the right development to that of the left). This is only possible where the left development travels outwards from right to left and the right development travels outwards from left to right (where the direction 'outwards' refers to 'along the perpendicular bisector of the sweep from one development to the other.) This would imply that the two developments are proceeding from positions in which their lower vertices have exchanged positions, which could only be possible if there were another intersection between the developments further upstream on the branch. This would mean that A or B were not in the lagoon's innermost collided pair. □

3.4.3 Repairing a single collision

A collision is *repaired* if one edge is removed from the cut-graph and another edge added in such a way that the original pair of crossed edges no longer intersect in the unfolding. A method for repairing a conflict is defined in Algorithm 3.

The design of Algorithm 3 is quite simple. The lowest crossing is identified and within it X, the first crossed edge. X determines P_X, the face which will not move. P_Y is then attached to P_X by unfolding the shortest path between them; there can now be no conflict between P_X and P_Y, as no cut-graph branch separates them.

The shortest path between two faces is the set of faces which contain the geodesic path between the center points of the two faces.

Caveat: In the author's implementation this algorithm is slightly modified: the shortest path is actually computed by minimizing the number of steps on the graph of face-to-face adjacency on the polyhedron, and the path chosen is not allowed to include the edge which had anchored P_Y to its parent. The motivation for this choice is discussed below.

Lemma 3.10 *If a collision between faces P_X and P_Y is repaired according to Algorithm 3 then there will no longer be any overlap between P_X and P_Y.*

Proof. If P_X and P_Y share an edge then this is trivially true.

If $\|L\| > 2$ then the algorithm will unfold the shortest path from P_X to P_Y into a strip of faces which contains a geodesic on the surface from P_X to P_Y and therefore contains a straight line in the unfolded plane. Thus the faces of

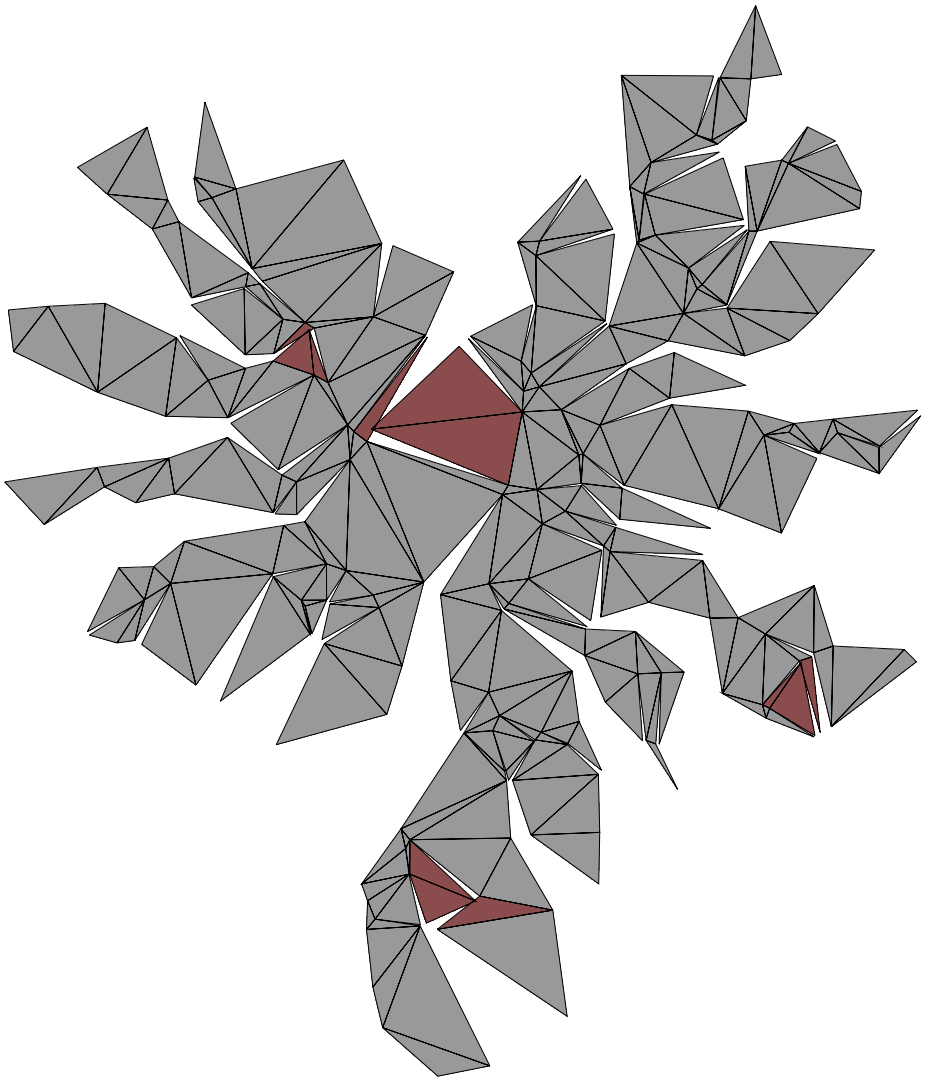
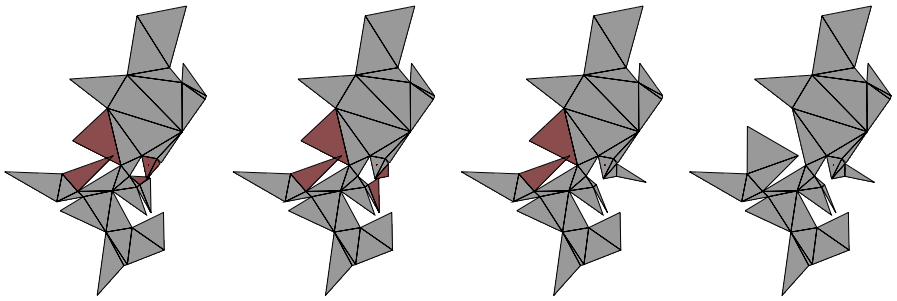


Figure 3.11: The unfolded net of a randomly-generated surface with 296 faces, which has four separate conflicts when unfolded by the Steepest Ascent Unfolder

Algorithm 3: Repairing a single collision in cut-graph T

1: Let edge X be the first crossed edge in the collision
2: Let edge Y be the edge furthest upstream which crosses X
3: Let face P_X be the polygon in the source polyhedron whose development contains the edge X
4: Let face P_Y be the polygon in the source polyhedron whose development contains the edge Y
5: Let $L = \{p_0 = P_X, p_1, \ldots, p_{n-1} = P_Y\}$ be the shortest path on the polyhedron from P_X to P_Y
6: **for** each $p_i \in L$, $i > 0$ **do**
7: Let e be the edge shared between p_{i-1} and p_i
8: Let g be the edge shared between p_i and the parent of p_i
9: **if** $e \in T$ **then**
10: Remove e from T
11: Add g to T
12: **end if**
13: **end for**



(a) (b) (c) (d)

Figure 3.12: The overlapped random unfolding of a random convex polyhedron of 38 faces is repaired by repeated application of Algorithm 3

the path cannot intersect. (Note, however, that no statement is made about whether or not the faces on the path intersect faces *not* on the path.) □

Corollary 3.11 *Every collision can be repaired.*

Proof. For every pair of faces, there exists a shortest path between them. If this path is unfolded as described above then the two faces cannot intersect. □

3.4.4 Application of the repair algorithm

It has now been shown that every collision can be repaired. Is this sufficient to remove all collisions from any overlapped net?

No. To repair a collision may very well introduce others. In experimental trials, the simple mechanism described thus far has been found to sometimes

Algorithm 4: Collision repair for randomized nets

```
1: Let U be a randomly-generated unfolding of the polyhedron
2: Let M be a map from nets to integers; M[u] is zero for any unfolded
   net u which has not been previously visited
3: Let F_0 be the first developed face
4: Let C be the center of F_0
5: while ∃ collisions in U do
6:    Let K be the collision in U closest to C
7:    if M[U] ≥ 4 then
8:       Return failure
9:    else if (M[U] ++) is not odd then
10:      Repair K by holding P_X fixed and moving P_Y
11:   else
12:      Repair K by holding P_Y fixed and moving P_X
13:   end if
14: end while
```

move a face across a cut-graph branch into a new conflict for which the only repair would be to move it back again in what would immediately become an infinite loop.

This is not to say that the algorithm does not have its successes. Given that there is no heuristic in the repair method which implies any sense of 'orientation' to the repairs, it has seem reasonable to apply this repair method to a randomly-generated unfolding of a randomly-generated polyhedron; a sample sequence of such unfoldings and repairs is shown in Figure 3.12. Simply applying Algorithm 3 repeatedly is sufficient to remove all collisions from the net shown.

Still, despite this encouraging result, infinite loops do occur. To avoid loops, a more globally-aware algorithm is needed. There are two reasonable approaches to such a problem:

1. Record nets which have already been visited and avoid them
2. Introduce heuristic knowledge from other successful unfolding algorithms

3.4.5 The first approach: repairing a randomly-generated net

The design of this algorithm may not be immediately clear, but it is surprisingly effective. The key insight here is that the choice to keep P_X fixed while moving P_Y in Algorithm 3 was essentially arbitrary. In fact there are two possible solutions to every collision: P_X may anchor P_Y or P_Y may anchor P_X.

Algorithm 4 repairs every conflict that it finds, knowing that in doing so it may move a face into a position in which the only repair for that face is to restore it to where it came from. Rather than falling into an infinite loop at this point, Algorithm 4 registers that this is a visit to a previously-encountered net and 'flips' the collision so as not to move the face which had previously been moved into conflict. Thus if a loop is found it simply tries to go the other way.

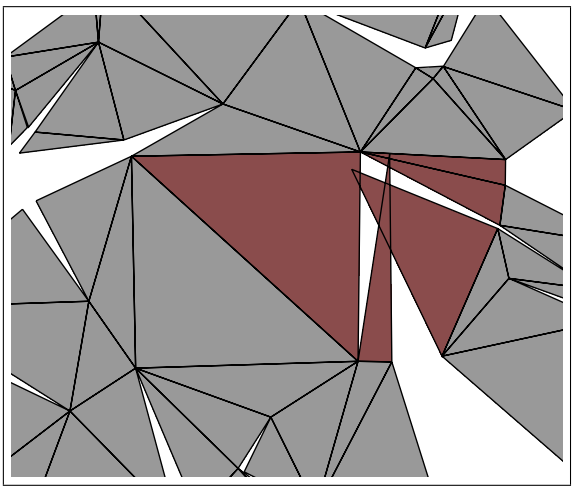


Figure 3.13: This detail from an overlapped random unfolding of a random convex polyhedron cannot be repaired by Algorithm 4. With two branches of the cut-graph competing to determine which direction is 'upstream' from the collision, the algorithm becomes caught in an infinite loop.

The odd choice of 4 for a termination test instead of 2 is driven by the fact that a 'state', if one were to think of each of the nets being generated as a node of a tree of nets to be searched, includes whether the last visit to the net was odd or even indexed. So if two faces are 'arguing' back and forth in a collision, it is not enough to try both directions of fix for only one of the possible nets; both possible nets for both placements of the faces must be explored. But if after both nets have tried to fix themselves with both possible solutions, none has worked, the algorithm must return failure.

Algorithm 4 is flexible, resilient to variable data... and is, essentially, a method to explore all possible repairs for a given initial unfolding. Infinite loops notwithstanding, it will eventually visit every possible solution of the net, and it does so in a guided exploration whose sole heuristic is to reduce the number of conflicts. The simple argument for Algorithm 4 is:

> Every repair removes (at least) one conflict and only some repairs introduce new ones. Therefore the total number of collisions in the net must decrease over time to zero.

Clearly, this is not a bullet-proof argument; but it expresses the spirit of the algorithm nicely.

Conjecture 3.12 *Algorithm 4 will repair the overlaps of any unfolding of any convex polyhedron.*

Rebuttal
Unfortunately a counterexample has been found. Figure 3.13 shows a detail from a randomly-generated unfolding which induces an infinite loop in Algorithm 4. The collision is oriented in such a way that the 'flip' of each possible repair

Algorithm 5: Collision repair for outward-facing nets

1: Let U be a Steepest Edge unfolding of the polyhedron
2: Let F_0 be the lowest face in the polyhedron
3: Let C be the center of F_0
4: **while** $\exists$ collisions in U **do**
5: Let K be the collision in U whose most upstream point of intersection is closest to C
6: Repair K
7: **end while**

(that is to say, the repair operation with the labels P_X, P_Y exchanged) is the unflipped repair from the other side of the conflict; the algorithm quickly reaches four flips and returns failure. □

3.4.6 The second approach: repairing an *outward-facing* net

Statistical analysis of the most successful unfolding algorithms has shown that one pattern of unfolding is more effective than all others: unfoldings in which the net, informally speaking, resembles a starburst. Schlickenrieder's Steepest Ascent algorithm[5] (Sch97, p.54), the discrete version of the Star Unfolding (p. 30), the author's Least Height Unfolder (p. 124), the alpha-beta rules (Section 3.3) and even breadth-first unfolding (p. 124) will consistently outperform algorithms which follow other patterns.

Let C be the center of the unfolding, the midpoint of the first developed face. Recall that the inner development of edge PQ is labeled P^IQ^I. A net is said to be *strictly outward-facing* if, for all edges $P \to Q$ where P is upstream from Q, $\|P^L - C\| < \|Q^L - C\|$ and $\|P^R - C\| < \|Q^R - C\|$. This requirement may be relaxed; a net is said to be simply *outward-facing* if, for every edge $P \to Q$ where P is upstream from Q, $\|P^I - C\| < \|Q^I - C\|$; no requirement is made on the outer development.

The author has found that the Steepest Edge Unfolder gives an outward-facing (and, in approximately 95% of the cases sampled, a strictly outward-facing)[6] net, which is sufficient for the repair algorithm. Using the Steepest Edge Unfolder to construct the 'seed' net for Algorithm 5 has had startlingly positive results.

One effect of performing repairs on an outward-facing net is that the repair cycle does not fall into an infinite loop. Intuitively, the reason for this is that each repair extends a cut-graph branch by adding at least one new cut edge to the downstream side of the 'lagoon'. The new cut edge has two developments, only one of which belongs to a face which has moved; the other edge belongs to an immobile face which is not a part of the current conflict. This means that if the newly-moved face has moved into an overlapping position, then it already has an edge whose dual is unconflicted, which will be the first crossed edge in

[5] Steepest Ascent: For each vertex, cut the edge to the vertex with the highest Y-value in the surrounding 1-ring

[6] *Caveat*: this claim was not tested for a statistically significant number of nets

the next repair. Therefore the face which was moved in the first repair will not move in the second, and so will not be returned to its former position.

The key idea of Algorithm 5 is that in an outward-facing net, the sides of a development point downstream and away from the unfolding's center. This means that there is a rough–not perfect, but sufficient–correlation between how far downstream a face falls on the cut-graph and how far from the center it will be when developed. On an outward-facing net when a repair moves a face from one side of a branch to the other, the face moves laterally, traveling roughly 'across the current' of the cut, with relatively minimal movement toward or away from the center of the unfolding. Since every face on the shortest path between P_X and P_Y must lie between the two in the flow of the cut, the repair algorithm ensures that no face further upstream than P_X and P_Y is impacted by the repair. Then the argument for Algorithm 5 is:

> Every repair removes (at least) one conflict and only some repairs introduce new ones, and no new collision will ever be introduced further upstream than P_X and P_Y. Therefore, even if the minimum distance from the center of the unfolding to the nearest collision may sometimes shrink for a single repair, it must increase in the long run, until there are no collisions left in the net.

Again, this is not a bullet-proof argument, but it expresses the spirit of the algorithm nicely.

And here is the amazing thing: *it works.* On simplicial convex polyhedra, it works every time. (So far.)

Algorithm 5 was tested on over 700,000 randomly-generated simplicial convex polyhedra of up to 300 vertices apiece, generated by taking the convex hulls of points randomly placed on a sphere. It was also tested against the banded icosahedron for $L = 0, 1, 2, 3$, and other known 'difficult' convex polyhedra. Even the counterexample to Algorithm 4 was unfolded without overlap, because the anchor-shaped cut-graph branch that defeated Algorithm 4 will not be generated in Algorithm 5.

Algorithm 5 has also been tested on over 75,000 randomly-generated non-simplicial convex polyhedra. Compared to the 700,000 simplicial tests, the author does not feel that this represents a sufficient sample, and so the claim that 'it works' will be restricted to simplicial polyhedra. Nonetheless, no counterexample has yet been found in either set of tests.

Algorithm 5 has displayed remarkable success at repairing a truly vast array of overlaps on simplicial and non-simplicial surfaces alike. It seems incredible that such a simple method could be so effective, but the testing data is undeniable.

3.4.7 Weaknesses in the argument

Collision Repair (Algorithm 5) is a method which can unfold every surface devised to date, but this *does not* constitute a proof that Collision Repair can unfold every convex polyhedron. The author deeply regrets that the proof remains incomplete.

Algorithm 4 would seem to be more powerful and flexible than Algorithm 5, but it can fail. Even if the current vulnerability to infinite loops were corrected,

the core design of the algorithm–a guided tree search which may exhaust all possible nets–implicitly implies that failure is possible. As such, while it may be more useful for the 'real world' (and, in particular, perhaps for non-convex surfaces) it cannot be used to prove that all convex polyhedra are unfoldable. In contrast, Algorithm 5 has no failure state.

The known flaws in the arguments are:

- A key lemma in support of Algorithm 3 would be to show formally that the unfolding of the path from P_X to P_Y cannot contain a self-intersection.
- A second key lemma in support of Algorithm 3 would be to show formally that no face in the path from P_X to P_Y will lie further upstream or downstream than P_X and P_Y.
- It is unclear *why* the initial net benefits from the outward-facing property. Theoretical support for this observed result would greatly benefit the proof.
- After a single repair, a net may lose the outward-facing property. Algorithm 5 does not address this explicitly yet in testing it has not been a problem. From this observation it can be conjectured the outward-facing property may be stronger than is necessary.
- A key lemma in support of Algorithm 5 would be to show formally that repair on an outward-facing net cannot fall into an infinite loop. The explanation given for why this is not the case is insufficient: it does not make clear how an outward-facing net behaves differently from a randomly-generated unfolding, an essential distinction.
- A second key lemma in support of Algorithm 5 would be to show formally that there is a bound on how much closer to the center a face being moved in a repair can travel.
- A useful corollary would then be to show that there is a bound on how much closer to the center a *second* face can be moved, if it is moved in response to the motion of a first face which has already been moved in a repair. There would seem, intuitively, to be an inward limit on such cascading motion.
- Although empirical testing is not required for the proof, it would lend further strength to the argument if non-simplicial convex polyhedra were to be tested as extensively as simplicial surfaces.

3.4.8 Experimental support

All figures in this section were generated by Algorithm 5.

Figure 3.14 shows a simple example: a net with two conflicts when unfolded by the Steepest Ascent Unfolder. Both conflicts were repaired.

Figure 3.15 shows the progressive repair of a randomly-generated non-simplicial convex polyhedron.

Figure 3.16 shows the progressive repair of the banded icosahedron.

Figure 3.17 shows the repaired unfolding of the *doubly-banded* icosahedron.

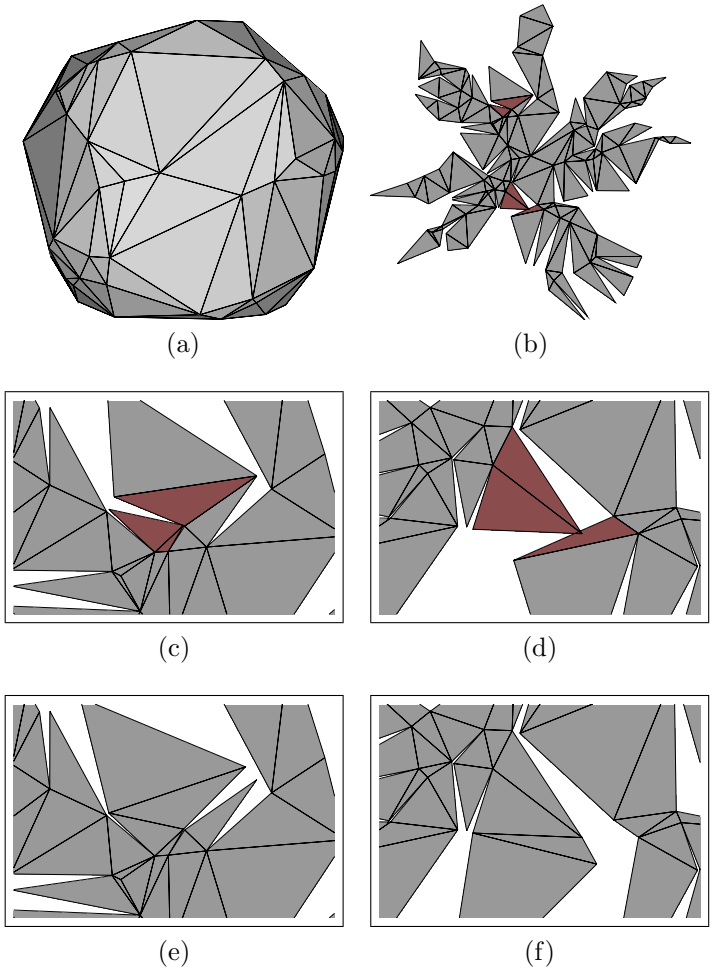


Figure 3.14: Repairing two simple conflicts. (a) The polyhedron (b) The unfolding, with collisions (c) Detail of the first collision (d) Detail of the second (e) The first collision, repaired (f) The second collision, repaired

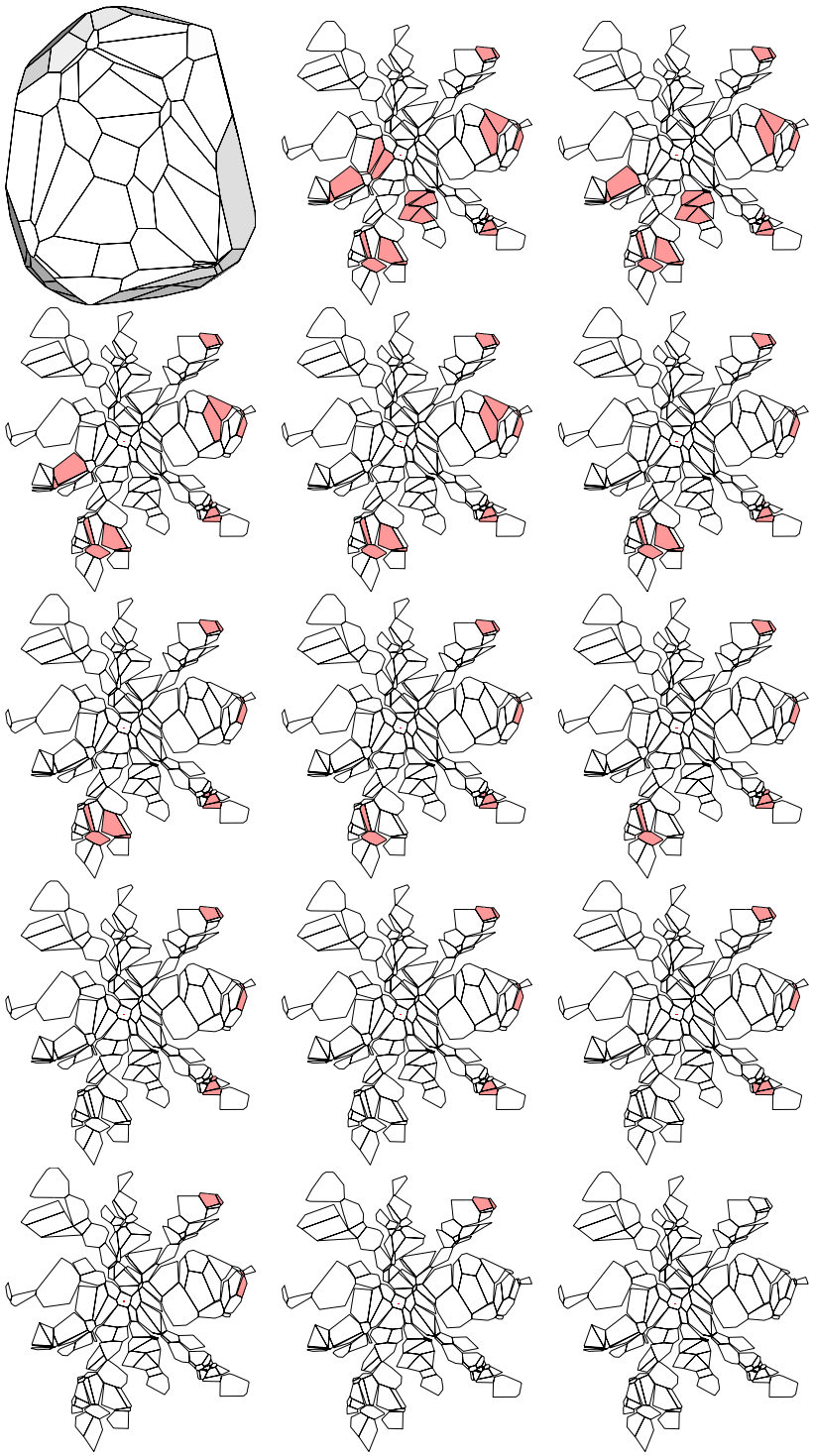


Figure 3.15: Repairing a randomly-generated convex polyhedron

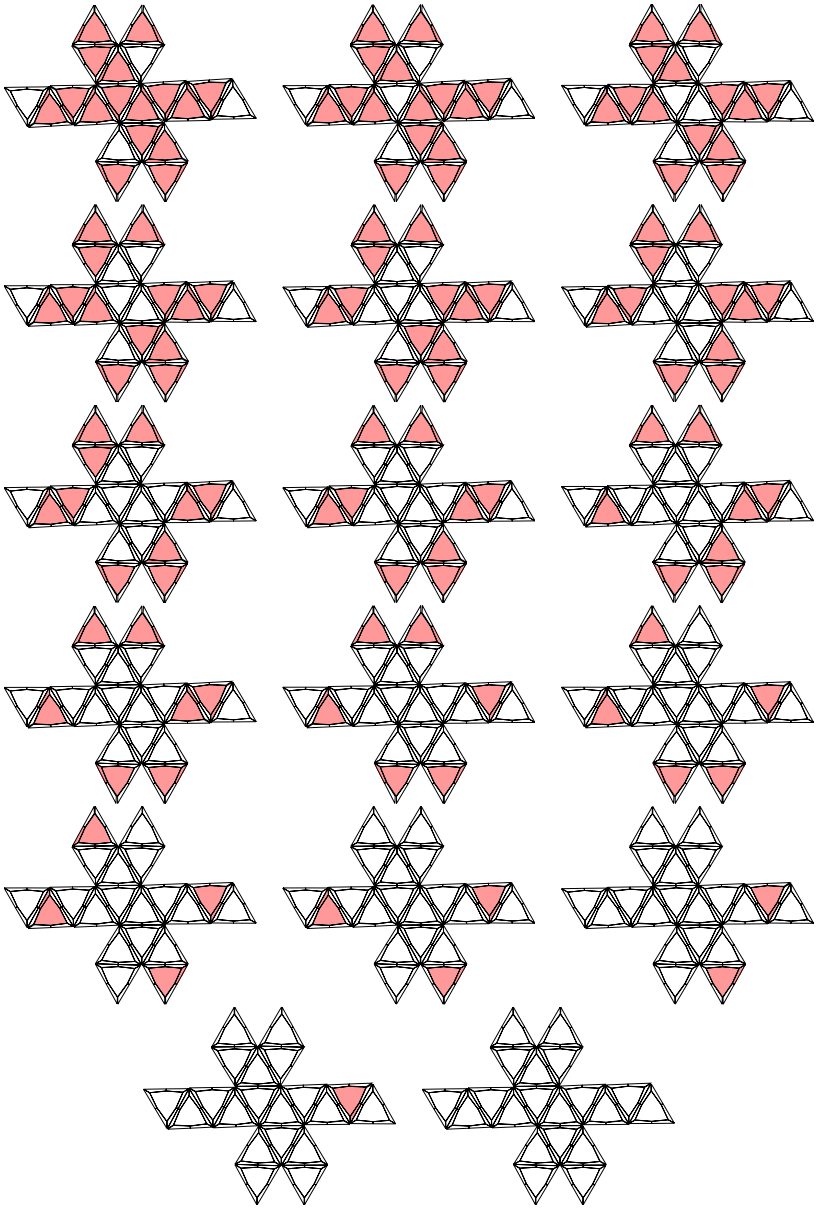


Figure 3.16: Repairing the banded icosahedron

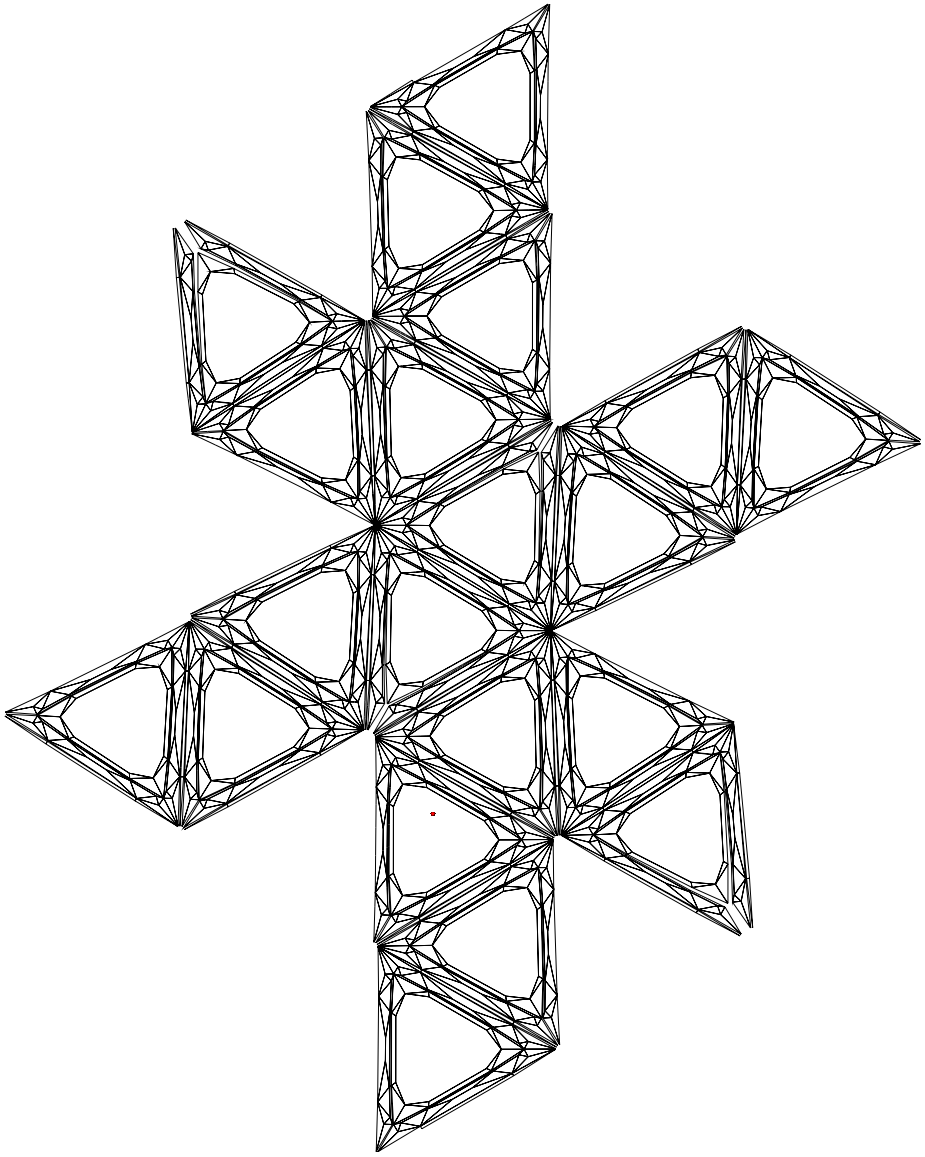


Figure 3.17: The doubly-banded icosahedron

3.5 Conclusions

The lower bound of undevelopable

The concept of the *smallest undevelopable convex polyhedron* has been introduced, and while it has been shown that the conjectured structure of the proof based on this polyhedron is untenable, the idea itself has not been shown to be invalid. The conjecture that the smallest undevelopable polyhedron can be shown not to have faces with three, four, or five sides–and hence cannot exist–remains open.

An intriguing side result of the consideration of this proof was Lemma 3.2, which shows that if the smallest undevelopable convex polyhedron P_0 does exist and has triangular faces which can be achieved through vertex truncation, then there are very strict requirements–so strict as to seem almost impossible to meet–on any polyhedron P_1 which could be truncated to P_0.

The alpha-beta rules

The alpha-beta rules consist of two lemmas and supporting corollaries which describe a partial ordering of the expansion of the vertices of a convex polyhedron. It is argued that if these rules are followed, any convex surface can be unfolded; it is also argued that these rules can always be followed. The proposed proof is incomplete, but it is supported by considerable statistical evidence: the alpha-beta rules are outperformed in experimental trials only by the Least Height Unfolder (Section 4.5.3) and the collision repair algorithms.

To complete the proof of the alpha-beta rules, much work remains. The backtracking issues which plague the software must be resolved if its data is to be viewed with any confidence. It will be necessary to extend the rules to cover the full range of values of α, and Lemma 3.7 must be made more robust in its proof that previous edges will be left uncrossed.

An alternative definition of V'_P has been suggested which incorporates $AD(P)$ into α_P. This is worthy of further investigation.

Collision repair

If it were possible to show that every set of collisions in any net of a convex surface can be repaired, then this would be a proof that all convex polyhedra are unfoldable.

It has been shown that every individual collision may be repaired. Two algorithms have been demonstrated which use this repairing operation with great success:

- Algorithm 4 explores all possible unfoldings, executing a tree search which seeks to minimize the number of overlaps, potentially visiting every possible of the polyhedron. Algorithm 4 is very complete but can fail, making it unsuited to a role in the proof: it is impossible to show that any algorithm which allows failure can repair every net.

- Algorithm 5 combines the Steepest Ascent Unfolder (Section 4.5.3) with collision repair to produce an algorithm which has successfully unfolded hundreds of thousands of randomly-generated simplicial convex polyhedra and tens of thousands of non-simplicial convex polyhedra, as well as all

> 'known to be hard' test cases. However, the author has been unable to argue conclusively why this particular combination of algorithms is so successful. Although a number of supporting ideas are advanced, the proof itself remains incomplete.

To complete the proof of collision repair, several formal demonstrations of supporting concepts are required, such as proofs that a shortest path on a polyhedron unfolds without self-intersection and that a face moved in a repair cannot be placed further upstream than the face which caused it to move. The connection between outward-facing unfoldings and the fact that a loop never occurs in Algorithm 5 must be clarified. This is a critical lemma, as it distinguishes the second approach from the first. Other supporting lemmas and demonstrations are also called for.

Still, the fact remains that to all available evidence, Algorithm 5 is a fully-functioning solution. To close the gap now between software implementation and mathematical proof may prove to be a lengthy task, but it is unquestionably worth the effort for future researchers in the field.

Chapter 4

Designing Algorithms for Convex and Non-Convex Polyhedra

4.1 Introduction

Within the Unfolding Problem lies the question,

> Given an arbitrary connected polyhedral surface, how quickly can a valid unfolding (or its impossibility) be determined?

This question addresses itself to both the convex polyhedra of Chapters 2 and 3 and to the much larger class of nonconvex polyhedra: surfaces with border, surfaces with bumps and holes, scanned surfaces (such as the cow model in Figure 4.1) and many more. The algorithms presented here cannot unfold every surface, for it is already known that some non-convex surfaces cannot be developed without overlap (such as the 'witch's hat' assembly, p. 13.) Instead this chapter explores the process by which an unfolding algorithm is designed and how these differing methods may be classified.

The algorithms discussed in this chapter are grouped, roughly, as follows:

- The 'brute force' unfolders, described in Section 4.4, are designed to operate independently of the convexity of the polyhedron upon which the algorithm executes.
- The 'progressive' unfolders, in Section 4.5, draw their function from heuristics–implicit or explicit–which draw on the features of convex surfaces. While the progressive unfolders can be run on non-convex polyhedra, to do so brings at best unpredictable results.
- The 'curvature-aware' unfolders are especially designed to operate on non-convex surfaces, and their performance suffers demonstrably on convex surfaces without border. Section 4.6 describes these algorithms, marking the first publication (to the author's knowledge) of algorithms specifically targeted at the general class of non-convex polyhedra.

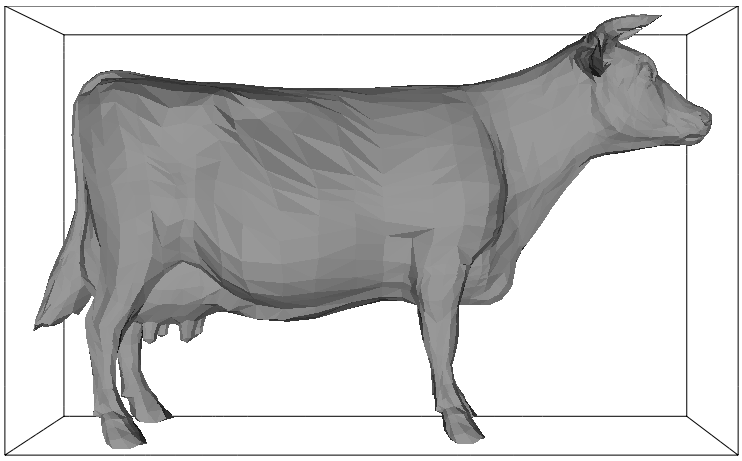


Figure 4.1: The polyhedral model of a cow, a nonconvex surface and one of the classic scanned models of contemporary computer graphics. The cow is often used in curvature-related research for its combination of broad and sharp regions of both positive and negative curvatures. It remains an open question whether this surface can be unfolded without overlap.

- The 'evolutionary' unfolders of Section 4.7 fall heterogeneously outside of the previous categories.

4.1.1 A question of graphs

The Unfolding Problem is, at heart, a quest for graphs. The polyhedron's 1-skeleton is a graph; the unfolded net is a graph; the cut-graph is (naturally) a graph. They all come together to give a single simple boolean report: does the unfolding self-intersect, or not? The graphs are constrained in their structure and configuration, restricted to rules that often can only be validated by measures which seem to lie beyond the state of the graphs themselves; the task of describing these rules in some manner which can be deterministically and procedurally evaluated has proved difficult, even daunting. But this is still, in the end, a quest for graphs.

The Unfolding Problems addressed in this dissertation fall broadly into two camps. The quest for a mathematical proof that all convex polyhedra are edge-unfoldable; and the quest for an algorithm which can edge-unfold any unfoldable surface (and reject those which are not unfoldable) in 'reasonable' time. This chapter deals with the latter, for if such an algorithm could be found it might also answer the former. Although with that said, the reader is cautioned that this is not necessarily the case: the existence of a method which can unfold all that is unfoldable, does not necessarily imply that everything (or even just everything convex) actually is. Still, there is cause for hope; a few of the approaches described here are resilient and robust in the face of highly variable data and should be taken as hopeful signs for future solutions.

To quote Malcolm Sabin[1], "An [unfolding] algorithm [would be] a constructive proof of the theorem that all convex polyhedra are unfoldable. Indeed, it is hard to see how any other kind of proof would work for the [...] mixture of topology and geometry that this problem has."

4.2 Categorizing Unfolding Methods

Unfolding algorithms, colloquially referred to as *Unfolders*, may be grouped into several different camps. Some methods optimistically select a candidate solution wholly-formed and evaluate it for success; others assemble a possible solution piece-by-piece, validating progressively; a third class seeks to evolve some (generally simpler) solution from a previous state toward a new solution.

Each of these approaches has both positive and negative traits. For example, validating a net to check for self-intersection is not a quick process, and while it can be accelerated with clever data structures it remains an operation bounded at best by the number of faces of the polyhedron. As such, the unfolders which create an entire cut-graph or unfolding net and then evaluate for validity may be described as 'optimistic' algorithms, for they defer this costly verification step until after computation is done. On the other hand, algorithms which build a net or cut-graph piecemeal, sequentially, will often evaluate their progress as they go, giving them immediate feedback on self-intersection and thereby opening the possibility of dynamic geometry-sensitive error-correction at the cost of a significant runtime overhead.

In describing an unfolding algorithm, the immediate output of the method may be described as a net of polygonal faces or as a cut-graph of broken edges. While these two forms of expression are exactly identical in terms of the data they convey, in practice they lead to notably different styles of algorithm development. It is often much more natural to approach a problem through addition rather than subtraction: that is, it is often more natural to view an unfolding as the serialized concatenation of individual faces rather than as a set of edge-removals.

For example, when a face is glued to another face, it is immediately clear if there is a conflict between the new face and the previous net; on the other hand if a cut edge is added to a cut-graph and self-intersection appears, the conflict induced may be at some far-distant edge across the unfolding where the relationship between cause and effect is much less clear. But then again, there are several schemes or methods such as local convexity (Section 2.6) and the alpha-beta rules (Section 3.3) which leverage the relationship between leaf-nodes of the cut-graph and dependent edges to reduce the probability of intersection. Thus both approaches have merits and downsides.

A third axis by which to distinguish families of unfolders is by their *decision classes*. Every unfolder implicitly chooses which edges to cut (or which edges to glue); different unfolders make these decisions based on different factors, and algorithms may be grouped accordingly. The scope of the decision class of the unfolder tends to determine the breadth of its output.

For example, the Iterative Unfolder (section 4.4.2) visits every combination of every possible gluing of every face in the polyhedron exactly once; it decides between solutions at the level of the unfolded net. The output of the Iterative

[1]Personal communications, 2007

Unfolder is a series of nets, including invalid nets with loops and disconnects. In contrast the Curvature Ordering Unfolder (section 4.6.2) makes choices as it is constructing its output, choosing the next face to glue to the unfolding by searching across the gradient of curvature. Thus its decision class is much more advanced and its output much more limited; in fact, its output is only a single unfolding which may or may not self-intersect. Occupying the middle ground between these two classes of decision coverage would be unfolders like the Least Height Unfolder (section 4.5.3,) which chooses the next face to glue by making a geometry-aware choice but which can be easily adapted to handle self-intersections encountered during unfolding, giving it an output of only a single unfolding but which reflects dynamic error correction.

The complete set of classes of unfolding methods is shown in Table 4.3. Table 4.1 shows a breakdown of each of the unfolding algorithms.

4.2.1 Schlickenrieder's Thesis

The lion's share of this chapter explores the series of unfolders designed and built by the author in the course of research. It is inevitable that such an exploration should bear a more than passing resemblance to the thesis of Wolfram Schlickenrieder (Sch97), whose evaluation of 34 different unfolders paved the way for significant later research. In fact, several of the algorithms described here are derived from or inspired by Schlickenrieder's algorithms. The author gratefully acknowledges Schlickenrieder's work.

Of the unfolding algorithms discussed here, the following four methods were first implemented by Schlickenrieder:

- Breadth-first (Sch97, p.39)
- Depth-first (Sch97, p.41)
- Star (Sch97, p.48)
- Steepest Ascent (Sch97, p.54)

Class of Construction	Brute Force	Create the whole net and evaluate
	Progressive	Create the net or cut-graph piece-by-piece and evaluate
	Evolutionary	Evolve the net from a previously-known-good state
Glue or Cut	Glue	Glue faces to available edge
	Cut	Add edges to growing cut-graph
Decision Class	Working face	Select next face from those adjacent to a 'working face'
	Working face set	Select next face from a working set (usually an advancing front)
	Any face	Select next face from all available faces
	Working edge	Select next edge from those adjacent to a 'working vertex'
	Working edge set	Select next edge from those in a working set (usually an advancing front)
	Any edge	Select next edge from all possible edges
	Working vertex	Select next vertex from a 'working vertex'
	Working vertex set	Select next vertex from a working set (usually an advancing front)
	Net	Whether or not the net self-intersects
	Cut-graph	Whether or not the net self-intersects
Decision Basis	Local geometry	Net or graph grows by immediate (local) geometry decisions
	Overall geometry	Net or graph grows by overall (global) geometry decisions
	Success	Unfolding either works or doesn't
	None	All unfoldings are considered, even if the first one found is intersection-free
Output Class	One unfolding by fixed root	Exactly one output, independent of geometry but dependent on an arbitrary choice of starting face or edge
	One unfolding	Exactly one output, independent of geometry and independent of starting place
	One error-corrected unfolding	Exactly one output but decisions are made during its generation which are dependent on geometry
	Series of unfoldings	One or more unfoldings, each evaluated in turn until one is successful, potentially yielding all unfoldings
	All possible unfoldings	All possible unfoldings
Error Tolerance	None	Output does not self-correct for geometry
	Adapt (partial)	Output self-corrects for local geometry
	Adapt (until successful)	Output self-corrects until successful (may extend beyond local geometry)

Table 4.1: Families of classification for unfolding algorithms

Unfolder	Method	Glue/Cut	Decision class	Decision Basis	Output	Error tolerance
Breadth-First ⊕	Progressive	Glue	Working face	Local geometry	One error-corrected unfolding	Adapt (partial)
Breadth/Depth Hybrid	Progressive	Glue	Working face	Local geometry	One error-corrected unfolding	Adapt (partial)
Convex Hull	Progressive	Glue	Working face set	Overall geometry	One error-corrected unfolding	Adapt (partial)
Curvature Ordering	Progressive	Glue	Working face set	Local geometry	One error-corrected unfolding	Adapt (partial)
Depth-First ⊕	Progressive	Glue	Working face set	Overall geometry	One error-corrected unfolding	Adapt (partial)
Least Height	Progressive	Glue	Working face set	Overall geometry	One unfolding	None
Path Minimizer	Progressive	Glue	Working face set	Overall geometry	One error-corrected unfolding	Adapt (partial)
Radius Minimizer	Progressive	Glue	Working face set	Overall geometry	One error-corrected unfolding	Adapt (partial)
Region	Progressive	Glue	Working face set	Local geometry	One unfolding	Adapt (partial)
Star ⊕	Progressive	Glue	Working face set	Local geometry	One unfolding by fixed root	None
Steepest Ascent ⊕	Progressive	Cut	Cut-graph	Overall geometry	One unfolding	None
Tracer	Progressive	Glue	Working face	Local geometry	One error-corrected unfolding	Adapt (partial)
Unravel	Progressive	Glue	Working face set	Overall geometry	One error-corrected unfolding	Adapt (partial)
Iterative Tree	Brute Force	Cut	Any face	None	Series of unfoldings	None
Random Cut	Brute Force	Cut	Any edge	None	Series of unfoldings	None
Precomputed Tree	Brute Force	Cut	Any face	None	All possible unfoldings	None
Spanned	Brute Force	Cut	Cut-graph	None	All possible unfoldings	None
Family Tree	Brute Force	Glue	Net	None	All possible unfoldings	None
Total	Brute Force	Glue	Net	None	All possible unfoldings (++)	None
Manual	—	Cut	—	—	Series of unfoldings	None
Truncation	Evolutionary	—	Working vertex set	Local geometry	One unfolding by fixed root	None
Error Correction	Evolutionary	Cut	Nearest Collision	Local geometry	One unfolding by fixed root	None
Anchorable Convex Hull	Progressive	Glue	Working face set	Local geometry	Series of unfoldings	Adapt (partial)
Alpha-Beta Rules	Progressive	Cut	Working vertex	Local geometry	Series of unfoldings	None
Local Convexity	Progressive	Cut	Working vertex set	Local geometry	Series of unfoldings	None

Table 4.2: Categorization of unfolders and conjectured algorithms (⊕ = implemented by Schlickenrieder)

Brute Force	**Progressive Heuristic**	**Progressive Geometric**	**Progressive Curvature**
Total	Breadth-First	Least Height	Curvature Ordering
Face Trees:	Depth-First	Steepest Ascent	Tracer
Precomputed	Depth / Breadth	Star	Region
Iterative		Unravel	
Spanned	**Evolutionary**	Convex Hull	**Other**
Family Tree	Truncation	Radius Minimizer	*Manual*
	Error Correction	Path Minimizer	*Random Cut*

Table 4.3: Families of unfolders

4.3 Heuristic Tree Search

4.3.1 Measuring self-intersection

It is natural to seek some measure by which one unfolding could be seen as more worthy of investigation than another. It would be ideal if there were some way to order the unfoldings of a polyhedron *before* they have been evaluated or even counted, as this would be an excellent decision class for an unfolding algorithm.

Demaine and O'Rourke introduce the concept of *overlap penetration*, which measures the interpenetration of two polygons. The degree of overlap is defined by "the ratio of the largest overlap penetration [ω_{max}] in [the unfolding] U to the diameter of the polyhedron P." (DO07, p. 308) Demaine and O'Rourke then pose Open Problem 22.2: to find "an algorithm that results in small overlap, for example, one that guarantees unfoldings with overlap penetration no worse than a constant fraction of ω_{max}."[2]

The natural question is, "if overlap penetration measures how 'badly' an unfolding has gone awry, why have known minimization methods (simulated annealing, etc) not been applied successfully to the unfolding problem?"

4.3.2 Discontinuous spaces and heuristic searches

At fault are the discontinuous natures of both the *decision space* and the *objective function*. These terms are drawn from optimization theory, which is the study of problems where the goal is the minimization of some scalar function. In the case of the unfolding problem, the 'decision space' is the choice of edge to cut or glue and the 'objective function' is the calculation of ω_{max}. Conventionally, optimization theory has focused on decision spaces and objective functions which are continuous; that is to say, small changes in the decision value should correspond to small changes in the objective function, allowing the algorithm to 'home in' on minimum values. Unfortunately, in the unfolding problem the decision space is discrete and the objective function is discontinuous; it is impossible to cut or glue less than one edge, and small changes in the cut-graph can result in large changes in ω_{max}.

[2] An odd yet valid counter-question would be to find the *worst* unfolding in terms of overlap penetration: to find an algorithm which produces the *greatest possible* ω_{max}. It is improbable that such an algorithm would be of use in the quest for valid unfoldings, but the question is still interesting in its own right.

Artificial intelligence techniques such as simulated annealing and genetic algorithms are better-suited to discontinuous decision spaces. Simulated annealing allows changes in the decision space which may seem to be 'bad' (i.e., in this case, to raise ω_{max}) so long as they are not *too* bad; the definition of 'too bad' is then gradually tightened, eventually prohibiting 'bad' moves, and the decision choice is then selected from the cut-graph or net with lowest ω_{max}. Unfortunately, a single edge change can provoke a dramatic rise or reduction in ω_{max} with little clear connection to the decision taken, and the discrete nature of the decision space means that sometimes the number of choices–good or bad–can be sorely limited.

As an example, consider a randomly-chosen polyhedron P of, say, 300 faces, with an average of four sides apiece. One might construct a decision tree G where the root node of G was a tree with one element: the bottommost face of P. Each child of the root of G would be a new tree with two elements: the bottommost face and one of its adjacent faces. Throughout G, each node of G would be a tree, consisting of the tree of its parent node plus one more face which had not yet been glued into the parent. With a bit of filtering to avoid duplicate nodes in G, one could construct a tree 300 levels deep where the bottommost row was a set of every possible unfolding of P. Every one of these possible unfoldings (a loose upper bound on their count would be 4^{300}, an amazingly large number) could have a different ω_{max}, and there would be no clear relationship whatsoever between the ω-scores of adjacent leaves in G. Furthermore, it would be impossible to evaluate ω_{max} for a given node without first having visited the node's parent in G. A semi-continuous method like simulated annealing would have no basis by which to choose a path from the root of G to the leaf of G which minimized ω_{max}.

To simply move blindly through the nodes of G, visiting them in some order determined only by the structure of G, could take as many as 4^{300} visits (a loose upper bound). A blind search taking no account of the data available at each node of G would clearly take far too long.

A better approach to such a discrete search is through *heuristic tree search*. A heuristic 'serves to aid discovery' (Nil71) and generally expresses some insight into the nature of the problem beyond the reach of formal expression. (The word itself derives from the Greek *heuriskein*, 'to find', the same root as "eureka".) Heuristic search leverages knowledge at each node of G to choose the next node to explore with better odds of ultimate success than a simple blind traversal.

In heuristic search, a list is held of *candidate nodes*, which initially consists of only a single element: the root of the search tree. A node is chosen for *expansion* from the list of candidate nodes and the children of that node are added to the list of candidates. If the node is the node sought–in this case, a complete net with minimal ω_{max}–then it is returned as a success.

Heuristic tree search optimizes around two principles:

1. *Pruning*: A good heuristic will reduce the number of nodes of G which would be considered for expansion before they are added to the candidate list.

2. *Ordering*: The heuristic algorithm can be intelligent in its choice of the next node from the candidate list to expand.

In the example given, a clever heuristic would realize that there would be

no point in adding to the candidate list any node of G where the partially-constructed unfolding already contained overlap (*pruning.*) As expansion progressed from the root of G, each time a newly-added face overlapped an existing face that branch of G would be instantly truncated. This simple filter is the basis for the *implicit error correction* used in several of the progressive unfolders (see Section 4.5, below.)

The second space of optimization offered by heuristic tree search, *ordering*, has to date proved to be more elusive. It is as yet unclear what ordering of gluing the faces of a net would most reduce overlap. In fact, it may well be that this is entirely the wrong question to ask, for as Lemma 4.1 below states, the final overlap state of a net is quite independent of the order in which the net was generated. Many ordering heuristics have been proposed, both by the author of this work and by other researchers; several are discussed here. The quest for an optimal heuristic remains open.

4.4 Brute Force Unfolders

The original motivation for exploring the 'brute force' unfolders was to construct an unfolder which *must*, eventually, find an overlap-free net if one existed. The plan was to find an upper bound on the complexity of the task of unfolding an arbitrary surface and to express that upper bound in code. This class of unfolders is described as *brute force* because they have no failure mode: each algorithm will carry on running until a solution is found–or, in some cases, until *every* solution is found–with no termination clause other than having reached the last possible net, even if the search involves quintillions of possibilities or more.

There is a second motivation for the development of the brute-force algorithms. If a polyhedron were found which the discoverer claimed had no overlap-free unfolding, it is difficult to conceive of a better proof than to exhaustively test every possible net for self-intersection. That there are so many possible nets is unfortunate but would not disprove the conjecture; it would merely prolong the verification of the claim.

Early attempts at building a brute-force unfolder were plagued by misconceptions and poor design choices, yet they have yielded several interesting results. Later unfolders improved dramatically on the performance of earlier designs. The unfolders described in this section include (in rough order of conception by the author) the *Total Unfolder*, which constructs every spanning tree of the faces of the polyhedron and does so once for every possible *ordering* of every possible tree; the *Precomputed* and *Iterative Unfolders*, which build procedural representations of all possible nets; the *Spanned Unfolder*, which constructs every spanning tree of the faces of the polyhedron; and the *Family Tree Unfolder*, which constructs every spanning tree through the incremental addition of faces and models each generation of spanning trees as a set of supersets of the previous generation.

It is worth noting that the brute force unfolders are tools which apply to non-convex and convex polyhedra alike. There is no heuristic preference in these algorithms, no weighting toward convexity.

4.4.1 The original brute force unfolder: the Total Unfolder

The goal of the *Total Unfolder* (Algorithm A.1) is to enumerate ALL the possible unfoldings of a mesh. This includes not only every possible way of traversing the faces of the mesh–that is, all the nets of possible unfoldings–but also to distinguish the different *orders* in which these unfolded nets could be laid down. The goal was an implementation where the unfolding that unfolded face A, then B off A, then C off A, would be distinct from the unfolding that unfolded A, then C off A, then B off A–even though the resulting nets were identical. In short, the goal was to enumerate

Every uniquely ordered traversal...
...of every spanning tree...
...that visits every face of the original polyhedron precisely once.

When the Total Unfolder was first designed, the author was operating under the (mis)impression that the order in which the faces were added to the unfolding was relevant; it was reasoned that an unfolding could be tested as it was laid down, and so the order in which it was built mattered. This was a mistake, which can be cleanly stated as follows:

Lemma 4.1 *The overlap of an unfolding is independent of the order of faces or edges by which that unfolding was originally calculated.*

Proof. The intersecting state of two line segments is independent of the order in which they are drawn. Similarly the intersection of two polygons in the plane is independent of the order in which they were added to the plane, depending only on their relative position and orientation[3]. □

Corollary 4.2 *The overlap of an unfolding is independent of choice of starting face.*

Thus the validity of an unfolding is independent of the order of its evaluation. Having noted this, it is worth noting as well that many validity tests can leverage orderings of the faces to optimize against unnecessary intersection tests.

The Total Unfolder implementation treats each face of the polyhedron P as an unordered collection of edges shared with another face. The algorithm builds a tree which represents every possible ordering of every possible unfolding; it does so by enumerating every possible traversal of the polyhedron's faces, from edge to edge.

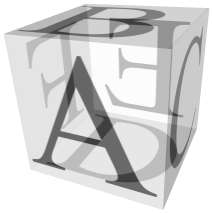
The Total Unfolder works with two distinct types of trees, for which informal nicknames are introduced:

Harvey A *Harvey* is a single partial unfolding of the polyhedron. Recall that a partial unfolding is a connected unfolded subset of the faces of a polyhedron (p.11.) Each distinct Harvey represents a single possible subnet of the faces of P. Each node of a Harvey is a face; each undirected edge of a Harvey represents the gluing of two faces in the unfolded net.

[3]In fact, one would be ill-advised to think of an unfolding as taking 'time' in any way. An algorithm generating an unfolding may take some amount of time to compute its answers but the answer itself contains no time. Each unfolding simply *is*, without anchor in the 4^{th} dimension.

Wayne A directed tree of Harveys is nicknamed *Wayne*. Each node of Wayne is a single partial unfolding. The edges of Wayne are directed edges; each edge $H \rightarrow H'$ represents the addition of a single face to Harvey H by gluing across a specific edge to create the new net H'. Each node H of Wayne is associated with a *perimeter set* which stores the list of edges which can be glued to H to produce H'.

Note that the nodes of Wayne are not distinct: multiple nodes in Wayne have the same value. However, each node is unique in that the sequence of directed edges which connects the root of Wayne to that node is unique. What is of interest in Wayne is how each node is reached, more so than the value stored at that node.



For example, to enumerate every possible ordering of every possible unfolding of the cube, one may begin with face A. From face A, the perimeter set contains four possible edges to glue: edges AB, AC, AD and AF. (Face E is inaccessible from A.) Suppose that the next face added is B, glued across the AB edge. Now the perimeter set of possible gluings available is AC, AD and AF–because these options remain open–and BC, BE and BF, because now B has been added to the growing Harvey.

With each new gluing, all but one of the edges of the newly-glued face are added to the perimeter set of the new node in Wayne. At the same time, all other ways of reaching that face can be removed from the perimeter set. For instance, were the next step to traverse AC, BC would be removed as it is no longer an option and CB or CA would not be added.

It is worth emphasizing that the edge-gluing sequence $A \leftarrow B$, $A \leftarrow C$ identifies a different node than the sequence $A \leftarrow C$, $A \leftarrow B$. Although the two partial unfoldings are absolutely identical, the paths by which they were constructed are not. The Total Unfolder generates all such paths.

To continue the example, consider Figure 4.2, which shows one possible sequence of edge-gluings to unfold a cube.

Step	*Face added*	*Perimeter set*	*Next edge*
(i)	A	$\{AB, AC, AD, AF\}$	AB
(ii)	B	$\{AC, AD, AF, BC, BE, BF\}$	AC
(iii)	C	$\{AD, AF, BE, BF, CD, CE\}$	AD
(iv)	D	$\{AF, BE, BF, CE, DE, DF\}$	AF
(v)	F	$\{BE, CE, DE, FE\}$	BE
(vi)	E	$\{\}$	

Figure 4.2 shows the gluing sequence $A \leftarrow B$, $A \leftarrow C$, $A \leftarrow D$, $A \leftarrow F$, $B \leftarrow E$. Wayne is depth six, the number of faces, which follows readily from the fact that each step of the algorithm down the tree eliminates one face.

By removing from the perimeter set those edges which now link to a face which has already been attached, the Total Unfolder avoids any risk of generating loops or disconnected subgraphs. (On a purely implementation-related note, detecting loops or disconnected subgraphs is an operation of at best $O(n)$ cost; this step of the Total Unfolder presents a significant saving to execution time.)

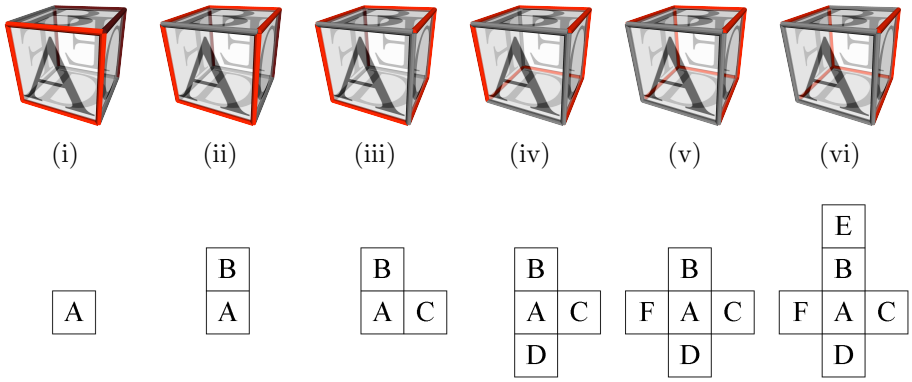


Figure 4.2: The progressive development of a cube unfolds a single path in the Total Unfolder

Each path through Wayne from root to leaf represents a single possible ordering of the generation of a Harvey, one possible unfolding of the original surface. It is now possible to begin to analyze the total number of paths through Wayne, i.e., the total number of unfoldings that would have to be tested in the quest for a valid unfolding with the Total Unfolder. Let:

- $\{ps\}_x$ denote the perimeter set of some node x in Wayne
- $\|\{ps\}\|_k$ denote the size of the perimeter sets at level k in Wayne, where the root is level 1
- n be the number of faces in P
- m be the average number of edges per face in the polyhedron P

For purposes of demonstration, it will be assumed that all faces of P have exactly m sides. Without this assumption, the mathematics quickly become unnecessarily complicated.

For a child node x at level k in Wayne, the perimeter set $\{ps\}_x$ contains at most $\|\{ps\}\|_{k-1} + (m-2)$ and at least $\|\{ps\}\|_{k-1} - m$ elements. To determine an upper bound on $\|\{ps\}\|_k$:

$$\begin{aligned} \|\{ps\}\|_1 &= m \\ \|\{ps\}\|_k &\leq \|\{ps\}\|_{k-1} + m - 2 \\ &\leq m + (k-1)(m-2) \end{aligned}$$

The *total number of children* F_k at each level k of Wayne can be bounded

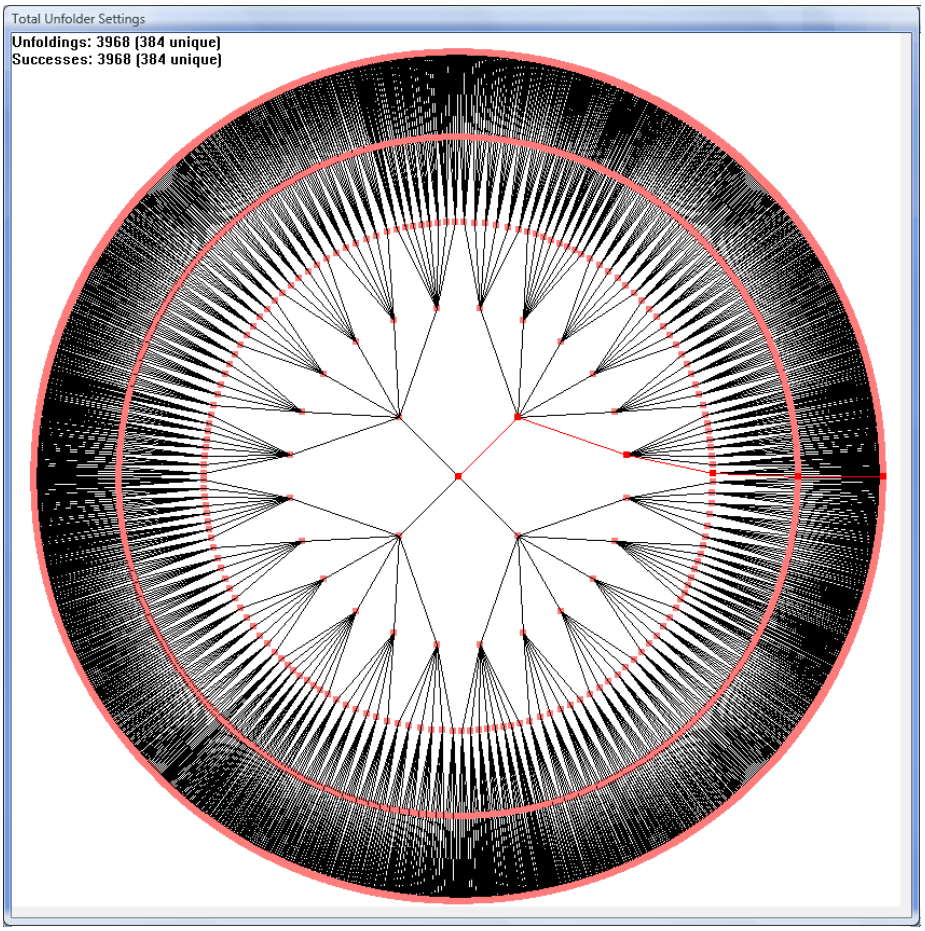


Figure 4.3: The Wayne tree for a cube. Each node in the graph is parent to all possible orderings of all possible unfoldings of each of the remaining nodes beneath it.

as follows:

$$\begin{aligned}
F_1 &= m \\
F_k &\leq \sum_{i=0}^{F_{k-1}} \|\{ps\}\|_{k-1} + (m-2) \\
&\leq \sum_{i=0}^{F_{k-1}} (k-1)(m-2) \\
&\leq (F_{k-1})(k-1)(m-2) \\
&\leq (F_{k-2})(k-2)(m-2)(k-1)(m-2) \\
&\leq (F_{k-3})(k-3)(m-2)(k-2)(m-2)(k-1)(m-2) \\
&\leq \ldots \\
&\leq m(m-2)^{(k-1)}(k-1)!
\end{aligned}$$

This indicates that the total number of unfoldings which could possibly be generated by the Total Unfolder is bounded above, regardless of the geometry of P, by

$$F_n \leq m(m-2)^{(n-1)}(n-1)!$$

This upper bound is actually quite loose. The mathematics given above describe the impossible worst-case scenario in which every step down Wayne increases the size of $\{ps\}$ by the maximum $(m-2)$ new nodes. However, this clearly can not be the case because, by the n^{th} level of Wayne, the number of faces remaining to be glued has dwindled to one. It is the last unattached face outside the level-$n-1$ Harvey, with its m possible gluing edges. Once the final face is glued, there are no further elements in the perimeter set:

$$\begin{aligned}
\|\{ps\}\|_{n-1} &= m \\
\|\{ps\}\|_n &= 0
\end{aligned}$$

Furthermore, since $\|\{ps\}\|_k - \|\{ps\}\|_{k+1}$ can be at most m at each step down the tree, it is known that

$$\|\{ps\}\|_{n-k} \leq km$$

This gives two concrete upper bounds:

$$\begin{aligned}
\|\{ps\}\|_k &\leq m + (k-1)(m-2) \\
\|\{ps\}\|_k &\leq (n-k)m
\end{aligned}$$

Solving for k, the intersection of these two linear functions[4] gives the index of r_k, the widest possible row of Wayne. From this it is known that

$$F_n \leq m(m-2)^{(r-1)}(r-1)!$$

[4]Solving for k:

$$\begin{aligned}
m + (k-1)(m-2) &= (n-k)m \\
m + km - m - 2k + 2 &= nm - km \\
(m-2)k + 2 &= (-m)k + nm \\
(2m-2)k &= nm - 2 \\
k &= (nm-2)/(2m-2)
\end{aligned}$$

and consequently

$$F_n \quad \sim \quad O((m^r)((r-1)!))$$

where $r = (nm-2)/(2m-2)$.

4.4.2 Second-generation brute force unfolders

The Total Unfolder, though interesting and strangely beautiful in its output (Figure 4.3), performs far more computation than is necessary. The *Precomputed* and *Iterative Unfolders* were both significant improvements over the Total Unfolder, abandoning the idea of ordering the generation of an unfolding and trading it for an improved running time of $O(K \times m^n)$, where K expresses the time taken to construct and test a single net[5]. Both schemes enumerate every possible unfolding by computing all possible combinations of uncut edges.

The Precomputed Unfolder

The Precomputed Unfolder computes all combinations of glued edges by building up an array of trees of nets (Algorithm A.2.) The final length of the array is m^n. The array is filled in in a series of waves, one per face. For each face in the polyhedron, all arrays previously created are duplicated, once for each edge of the face, and each duplicate is extended with the face's neighbor across the corresponding edge. This form of search will inevitably create loops in the nets being constructed, so if a net is created that contains a loop it is immediately discarded. When a net is discarded, its entry in the array is zeroed (line 22) and remains zero through all subsequent faces, allowing the Precomputed Folder to avoid redundantly processing looping graphs of the polyhedron's faces. To ensure that each net is a connected graph, faces are initially sorted into a list in which each face is either first or shares an edge with a face earlier in the list.

The Precomputed Unfolder has a running time of $O(K \times m^n)$. However, many of the positions it visits can be rejected trivially, which significantly reduces that upper bound. It consumes $O(n \times m^n)$ space, constructing a new tree for every (non-looping) combination of edges.

The Iterative Unfolder

The Iterative Unfolder follows a different implementation toward an almost identical destination to its peers. To address the tremendous space demands of the Precomputed Unfolder and to allow user interactivity during computation while still generating every possible unfolding net, the Iterative Unfolder describes a single unfolding of a polyhedron P with n faces, each of which has m edges, as a length-n, base-m integer. Each 'digit' in the integer has valid values from 0 to $m-1$, representing by which edge that face is glued to its parent face in

[5]The precise mechanism by which a net is constructed, if that construction is not a natural side-effect of the algorithm, is treated as a non-factor here. Construction is generally an $O(n)$ process. Testing for collision, on the other hand, can be optimized within the evaluation of a single net through the use of a *BSP Tree*, a data structure which accelerates spatial testing of a function such as polygon overlap. The BSP tree is discussed in more detail in Appendix B.7.3. The particular method of net generation and the specific technology of intersection testing are not directly germane to the unfolding algorithms discussed here, and will be disregarded henceforth as 'solved' problems.

the unfolded net (a static choice of root face is assumed.) The main processing loop is then quite simple: the current index is incremented (in base m) and the new unfolding tested for loops and disconnects; while loops or disconnects are found, the index is incremented again. When the integer overflows, computation is complete.

The integer representation described is clearly a function of implementation, as no edge of a face of a cube comes 'first'. However, an arbitrary ordering function may be assigned with no loss of generality, such as lexicographic order (p. 9.) Given any consistent ordering and some mechanism that always selected the same root face upon which to center an unfolding (such as 'least height' or similar) then an entire unfolding could be expressed in at most $n\lceil \log_2 m \rceil$ bits.

The Iterative Unfolder might at first seem to have a worst-case running time of $O(2^{n\lceil \log_2 m \rceil})$ as it evaluates every value encodeable in its bit-compression, but thankfully it actually outperforms this pessimistic estimate. The Iterative Unfolder has a running time of $O(K \times m^n)$, evaluating only those bit-patterns which constitute valid combinations of the edges of P. Unlike the Precomputed Unfolder it cannot trivially discard looped or disconnected trees early and thereby avoid processing them later, which negatively impacts its practical running time. This is balanced by its significantly better use of space, storing most passive data in a highly compressed form, and its interactivity. The user begins to see results from the Iterative Unfolder almost instantly after processing begins.

4.4.3 Third-generation brute force unfolders

The second generation of brute force unfolders were designed to construct every possible net of the polyhedron by enumerating every possible combination of face gluings. Both algorithms suffered tremendously from the overhead of catching and filtering graphs with loops and disconnected graphs from their final list of trees.

The third generation of brute force unfolders uses a more advanced method: evaluating every spanning tree of the polyhedron's graph of face-to-face connectivity. A number of algorithms exist for finding all the spanning trees of a graph; recent work by Kapoor, Kumar and Ramesh (KR00) runs in $\mathrm{O}(N \log V + V^2\alpha(V,V) + VE)$ time where the directed graph has V vertices, E edges and N spanning trees and α is the inverse Ackerman function.

The Spanned Unfolder

The *Spanned Unfolder* (Algorithm A.4) converts the input polyhedron P to a graph G of the face-to-face connectivity of P, generates the set of all spanning trees of G, and returns that set as the result. Recall that the lower bound on the number N of spanning trees of $G = (V, E)$ is $2^{O(\sqrt{\|E\| - \|V\|})}$ (RT75).

Lemma 4.3 *Given a polyhedron P having F faces, $E/2$ face-to-face connections and N spanning trees of the graph of all face-to-face connections, where K is the average time to test a single net of P for overlap, a non-self-intersecting unfolding of P may be found or determined not to exist in $O(K \times (N \log F + F^2\alpha(F,F) + FE))$ time.*

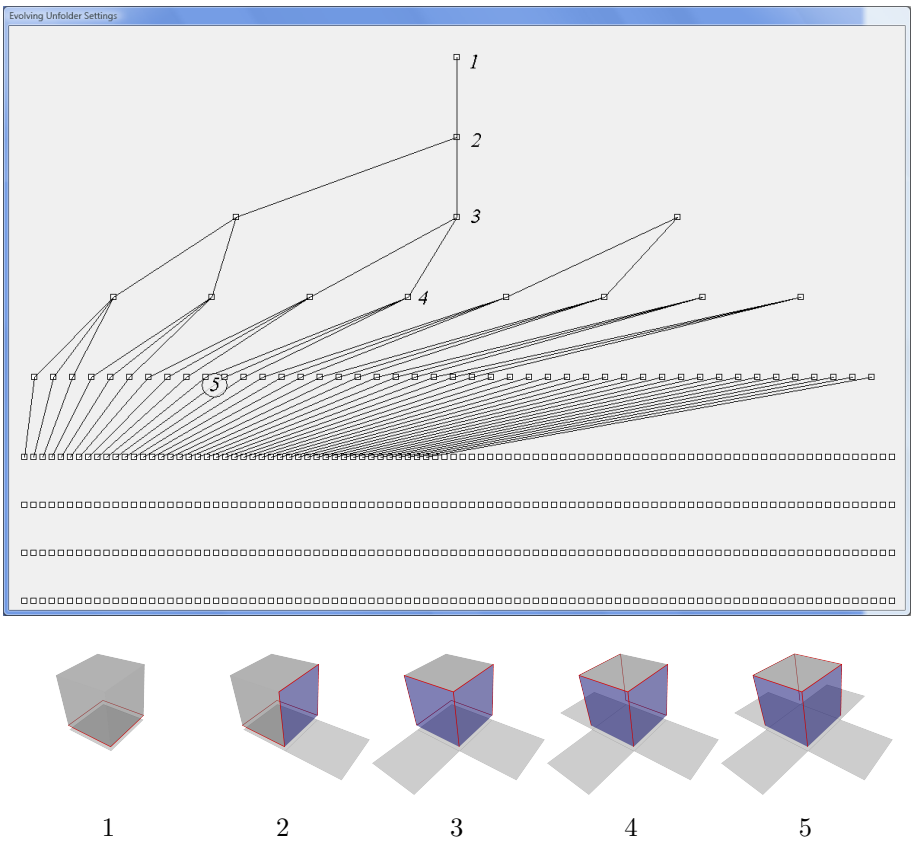


Figure 4.4: The 'family tree' of an unfolding of a cube showing how for a given ordering of the faces of the polyhedron, in constructing partial polyhedra with border there is an inheritance relationship between some–but not all–of the unfoldings of each generation.

Proof. By existence of the Spanned Unfolder and Kapoor et al.'s algorithm. Note that E is divided by two in the lemma to support converting every face-to-face adjacency to two directed edges. □

The Family Tree Unfolder

Consider an n-faced polyhedron P_n, possibly with border. Let G_n be the graph of face-to-face connectivity of P_n. Let P_{n-1} be the polyhedron P_n with a single face removed, and define G_{n-1} in like manner. The *Family Tree Unfolder* (Algorithm A.5) was inspired by the insight that the set of all spanning trees of G_n must necessarily include a set of trees which were themselves supersets of each of the spanning trees of G_{n-1}. That is to say, all of the possible unfoldings of P_{n-1} are contained within all of the possible unfoldings of P_n.

The Family Tree Unfolder is not, strictly speaking, a new Unfolder. Encapsulated within the FTU is the core of the Spanned Unfolder, discussed above. Instead, the FTU is a visualization tool: it images the relationship between

successive steps of the unfolding of a polyhedron by modeling each step as a superset of the set of all spanning trees of the previous step. The resulting imagery is quite intriguing, forming a disconnected graph in which each node is a partial unfolding and each edge indicates a subset relationship (Figure 4.4.)

The Family Tree Unfolder first sorts the faces of the polyhedron into a list using the same sort as the Precomputed Unfolder, producing a list in which every face is guaranteed to share an edge with another face which precedes it in the list. From this list of n faces, n polyhedra with border are generated: each polyhedron P_i consists of only the first i faces of P. The Family Tree Unfolder runs the algorithm of the Spanned Unfolder on each P_i, generating the set $\{U_i\}$ of unfoldings of P_i. Then for each spanning subtree u in $\{U_i\}$, the Family Tree Unfolder searches $\{U_{i-1}\}$ seeking the one spanning tree u' which is a proper subset of u. This creates a 'family tree' of unfoldings (Figure 4.4) showing how each unfolding either could be built from the previous generation of unfoldings (less one face) or is an entirely new net which had not been visited before.

4.5 Progressive Unfolders for Convex Polyhedra

A *progressive unfolder* uses tree search techniques[6] to construct an unfolding. For most of the Unfolders described in this section, the graph G is the graph of connectivity between the faces of the polyhedron P; each node of G is a face of P and each edge of G represents a shared polyhedral edge between the two faces of P. As G is searched, an unfolded net of P can be *progressively* constructed.

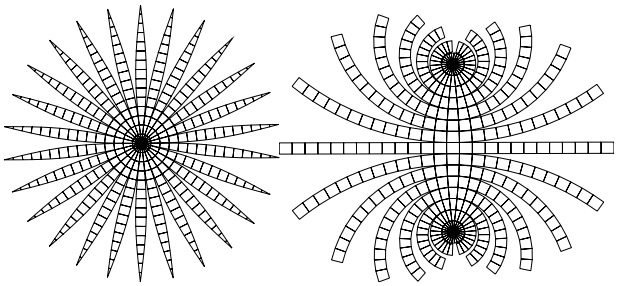
Two exceptions to this definition of G are the Steepest Ascent Unfolder and the Star Unfolder, described below. These algorithms construct a cut-graph edge-by-edge instead of a net face-by-face.

Blind search

Recall that heuristic tree search uses two forms of optimization, *pruning* and *ordering*, to direct its output toward the goal. The first generation of progressive unfolders apply pruning techniques to filter overlapped partial unfoldings and use blind searches (depth-first and breadth-first) to construct the unfolded net from the remaining nodes. This pruning method, in which the partial unfolding is constructed as the net is calculated and each face considered for gluing to the partial unfolding is evaluated for overlap before addition, is called *implicit error correction* because it acts to passively guide an otherwise blind search.

It should be noted that implicit error correction is much more readily implemented for unfolders which progressively assemble the net of the unfolding, rather than the cut-graph. As a net is assembled it is relatively easy to test for overlap. As a cut-graph is assembled, overlap is more difficult to predict. Thus a blind breadth-first sort, with pruning, will often outperform the more advanced Star Unfolder which cannot use this technique. In fact breadth-first sort ranks among the five highest-performing unfolding algorithms on randomly-generated simplicial convex polyhedra (see Section 4.9, 'Comparison of Results'.)

[6] The tree to be searched is actually a graph, but a simple pruning by prior visit allows the graph to be visited without cycles.



(a) (b)

Figure 4.5: The breadth-first unfolding of a parametric sphere, shown with the edges of each node of the search tree (each face of P) ordered in counterclockwise order from (a) the lowest first, for lexicographic order; and (b) by order of instantiation in the test software

Geometric heuristic

The second generation of progressive unfolders uses simple geometric heuristics to order faces before gluing. Using such information as least height and shortest path, these unfolders perform well on convex surfaces but generally more poorly on nonconvex surfaces, which shows that the heuristics chosen emphasize the traits of convex polyhedra.

Constrained output

The third generation of progressive unfolders works against *output constraints* instead of input (source polyhedron) criteria. Such constraints include minimizing the total area of the convex hull, minimizing the maximum length of paths between the centers of connected faces in the plane, and minimizing the total radius of the unfolding.

4.5.1 Ordering the graph

In building the unfolded net by gluing edges in a blind search, an ordering must be chosen *a priori* to order the children of each node in G (the list of neighbors of each face in P.) One such ordering is lexicographic ordering (p. 9.) Another is by simple order of instantiation of the faces. Early algorithms chose an ordering based solely on the order of construction of the original polyhedron, which yielded irreproducible results which were entirely a function of the implementation. The lack of clarity arises from the fact that neither the 1-skeleton of a polyhedron, nor the graph of its face-to-face connectivity, has an implicit ordering.

The author shifted in time to a lexicographic sort, which is an ordering independent of implementation. This led to an increase in stability of all algorithms so modified. The degree to which some algorithms were impacted by such a small change was sometimes quite striking; the before-and-after contrast is well-demonstrated by the Breadth-First Unfolder (described below) in Figure 4.5.

4.5.2 Blind search progressive unfolders

Breadth-first Unfolder

Breadth-first Unfolder (Algorithm A.6; (Sch97, rule 1, p.39)): Performs a breadth-first traversal of G, gluing faces to the unfolding across the edge on which they are first encountered. Implemented by maintaining a queue of faces 'to be done' and appending the neighbors of newly-unfolded faces to the tail of the queue.

Depth-first Unfolder

Depth-first Unfolder (Algorithm A.7; (Sch97, rule 2, p.41)): Performs a depth-first traversal of G, gluing faces to the unfolding across the edge on which they are first encountered. Implemented by maintaining a queue of faces 'to be done' and pushing the first neighbor of each newly-unfolded face onto the head of the queue. Note that for simplicial convex polyhedra of up to roughly 60 faces the Depth-first Unfolder actually performs *worse* than every other algorithm discussed in this chapter, including completely randomized search.

Breadth/Depth Hybrid Unfolder

Breadth/Depth Hybrid Unfolder (Algorithm A.8): Performs a breadth-first traversal of G but visits all children of a node before expanding the siblings of the node. This 'hybrid' unfolder was originally designed in the expectation of testing depth-limited search and comparing the relative performance of different limits on depth; the plan was to chart average overlap vs. depth limit on a series of random polyhedra. This line of research was eventually set aside and the breadth/depth hybrid algorithm remains fixed at a depth-limited search of depth 1, embedded within a breadth-first model.

4.5.3 Geometric heuristic progressive unfolders

Least Height Unfolder

Least Height Unfolder (Algorithm A.9): Unfolds the polyhedron by selecting a starting face and then repeatedly gluing the lowest face in the border of the unfolding to its lowest neighbor. The resulting geometry tends to to stay close to curves which are locally geodesic on the surface, giving excellent results on convex surfaces but ill-suited to nonconvex regions. This method uses implicit error correction and produces nets which are very similar to Schlickenrieder's Steepest Ascent (see below) but with lower overlap rate. When tested against large suites of randomly-generated convex polyhedra the Least Height Unfolder outperforms almost every other algorithm discussed in this chapter. It takes second place only to Collision Repair.

Star Unfolder

Star Unfolder (Algorithm A.10; (Sch97, rule 4, p.48)): Uses Dijkstra's $O(n^3)$ shortest-path algorithm to find the shortest paths from a point on a chosen face to every vertex (see Section 2.3.) These paths (with the intra-face cuts removed, of course) become the cut-graph.

Steepest Ascent Unfolder

Steepest Ascent Unfolder (Algorithm A.11; (Sch97, rule 7, p.54)): Implements Schlickenrieder's Steepest-Ascent algorithm, in which for every vertex v a single incident edge is cut from v to the highest (greatest y co-ordinate) vertex in the 1-ring of v.

Unravel Unfolder

Unravel Unfolder (Algorithm A.12): The Unravel Unfolder was designed to test the unfoldability of *coolinoids* (Chapter 5.) The Unravel Unfolder finds a Hamiltonian path on the surface, working from the face with the least y value to the face with the greatest. The nets produced often strongly resemble those generated by a depth-first search. Two versions of the Unravel Unfolder have been developed. The one tested here adds a second error-handling pass which attempts to find another edge to unfold any faces that could not be unfolded in the first path, which is allowed to break the 'Hamiltonicity' of the path.

4.5.4 Progressive unfolders with constrained output

Convex Hull Unfolder

Convex Hull Unfolder (Algorithm A.13): Chooses the next step in the unfolding which will minimize the area of the convex hull. The resulting cut-graphs are often chaotic and seem partially random in structure as the convex hull loses symmetry. This method seeks the local minima of hull area; a global search would be significantly more expensive. The Convex Hull Unfolder performs surprisingly well in testing, given that it would seem at first glance to be an entirely irrelevant feature of the unfolding. Some insight into why this should be so may be gained from the following observation: 2-local overlap requires that there be an open gap inside the unfolded net, the 'lagoon' of Section 3.4. Any algorithm which seeks to minimize the area of a convex hull will tend to fill in these gaps before adding later faces, which will often (though by no means every time) glue a face inside the gap instead of gluing it into an overlapped position.

One related side-avenue of research which was not fully explored was a variant of the Convex Hull Unfolder which sought to *minimize* the amount of empty space in the convex hull. The results seen were promising, and seemed to support the theory about gaps being filled. For larger nets, however, the positive effects of the change vanished and performance began to suffer, so that variant of the Unfolder was abandoned.

Path Minimizer Unfolder

Path Minimizer Unfolder (Algorithm A.14): Chooses the next step in the unfolding which will minimize the length of the longest path in the unfolding. Length is calculated as total unfolded center-to-center distance in the plane. This naturally encourages a close-packed, radial arrangement.

Radius Minimizer Unfolder

Radius Minimizer Unfolder (Algorithm A.15): Chooses the next step in the unfolding which will minimize the greatest radius of the unfolding. Radius is calculated as center-to-center distance from each face in the plane to the seed face. Unfoldings again tend to be radial but with more variation further from the center of the unfolding than in the Path Minimizer.

4.6 Curvature Unfolders for Non-Convex Polyhedra

A third suite of unfolders has explored applying heuristic approaches to curvature data to specifically target non-convex surfaces.

The first generation of non-convex unfolders, the Curvature Ordering Unfolder ('COU'), uses curvature data to sort an advancing wavefront of possible faces. A second generation extended this idea to incorporate explicit traversal of faces by curvature gradient. The third generation of curvature-aware unfolders has sought to apply the COU design to multiple regions in parallel.

The philosophy of the Curvature family of Unfolders is to build star-like unfoldings in regions of positive curvature in the anticipation that by the time negative curvature is encountered, the faces will have 'fanned out' far enough apart that they cannot self-intersect. For some nonconvex surfaces which would otherwise be difficult to unfold this philosophy can work well.

4.6.1 Local curvature and angle deficit

Before discussing the curvature-aware family of Unfolders, a few words are appropriate about angle deficit, the means by which that curvature is determined.

In one sense, angle deficit is the perfect measure of curvature at a vertex. In building a cut-graph it is unambiguously true that a vertex with negative angle deficit must be cut at least twice, where a positive vertex need only be cut once.

In another sense, angle deficit is a *terrible* measure of surface curvature. A strategy built on the concept of 'regions of positive curvature' cannot depend on finding a reasonable description of such regions from the discontinuous sampling of angle deficits at vertices. Any per-vertex trait can be undermined by closely-packed sliver polygons; for example, consider the Utah teapot (Figure 4.6.) The Utah teapot model has a fine line of slim rectangles lining the outer edge of the lid. In a curvature map of the teapot one sees that the body of the teapot curls toward the edge of the top and moves toward strong negative angle deficit at each vertex around the lip, but then the top of the teapot itself has no negative (red) on it: the strip of rectangles introduces an extra line of vertices and the negative curvature information is lost.

Thus while angle deficit is still an essential tool in building the unfoldings described here, it is not a sufficient replacement for more advanced feature detection techniques.

A second point worthy of discussion when considering angle deficit as a heuristic for unfolding stems from Polthier's observations (p. 14.) Angle deficit is usually not calculated as a simple sum of face angles. That is to say, while the simplest definition of the angle deficit of a vertex v is $AD(v) = 2\pi - \sum \alpha(f, v)$

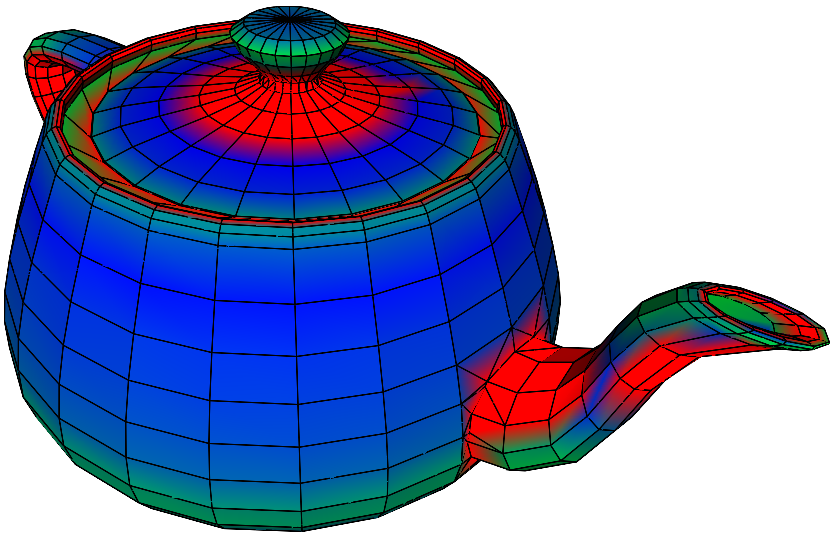


Figure 4.6: A teapot, shaded by angle deficit with negative regions emphasized in red. A fine ring of slim polygons around the rim of the teapot cause difficulty when angle deficit is used to find curvature for larger regions of the surface.

(see p. 5), it is common in research to use the smoother function $AD(v) = 2\pi - \sum \alpha(f, v)/A_{Mixed}$ where A_{Mixed} is the Voronoi area of v[7] However, in unfolding, the angle deficit seems to be a primarily binary determining factor: if it is greater than or equal to zero then the cut-graph may have a leaf at v; if it is not, then the cut-graph may not. Thus the version of $AD(v)$ used in this work is the simpler equation.

4.6.2 Curvature Ordering Unfolder

The *Curvature Ordering Unfolder* (COU) (Figures 4.7, 4.8, 4.9; Algorithm A.16) is a curvature-aware unfolder. Every face is assigned an "average curvature", the average of the curvatures of the vertices of the face. (Vertices on boundaries are arbitrarily assigned -2π.) The face with the highest curvature is then chosen as the root of the net and all subsequent unfolding flows from there. At every pass, the next face to glue onto the net is chosen from the boundary set of the current net by searching the boundary for the face with the greatest curvature. This leads to a tree whose branches and leaves decrease in curvature steadily as they get further from the original root, with the most negative of leaves toward the extremities.

The design is that negative regions are to be broken up and distributed across the unfolded plane. As negative curvature on a polyhedron is usually found fairly far from the positive regions, rooting a partial unfolding amongst the most positive faces should ensure that faces with negative curvature are not left together to overlap. Another way of looking at this would be to say that the cut-graph must begin at the most negative vertices and radiate outwards from

[7]For details on smooth discrete curvature maps and pseudo-code for finding A_{Mixed} for both acute and obtuse angles the reader is referred to (MDSB02, p.10,12).

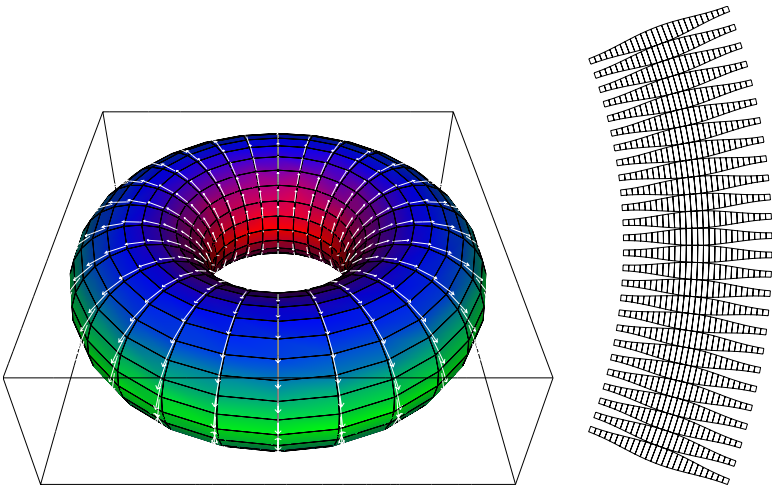


Figure 4.7: A torus, colored by curvature and unfolded by the Curvature Ordering Unfolder

there; Curvature Ordering tries to express that key concept, although because it is a gluing-oriented algorithm the cut-graph is less clearly controlled.

In practice, this tack works quite well on some surfaces but not on others. If the root face first chosen has positive curvature, then this approach splits apart zones of negative curvature by laying down the regions that surround the negative curvature on different strips of joined faces. But this trend is not guaranteed, because the "most positive" region is not always as far as possible from the "most negative" region, if distance is measured by the number of faces and not their breadth. There is no support in this approach for multiple areas of negative curvature or for finding the best root in a region of uniform positive curvature. If the entire surface is a region of negative curvature (such as the spun parabola or *coolinoid* (see Chapter 5) then this approach splits the regions that are more negative than others, but must leave some negative region around the root untouched and therefore ununfoldable.

The real failing here is in the ordering heuristic chosen. The magnitude of local curvature is not actually relevant; what is important is to split the regions of negative curvature as widely apart as possible. In other words, to root the cut-graph in the negative regions. This argument would seem to call for a cut-graph-oriented implementation instead of a face-gluing implementation.

Ordering the faces based on curvature, and selecting only from the most positive faces in the current boundary set, are both false assumptions. First of all, the next-most-positive region may be connected to the current region by a band of negative curvature. This approach will not cross that band until it has exhausted all of the positive faces nearby, but multiple positive regions should all be equally deserving of being roots of the unfolding tree. Second, ordering based on curvature itself is misleading because it implies a more continuous view of the surface than is actually correct. In the limit surface approximated by the polygonal mesh, curvature may change smoothly so that positive regions are clearly separated from negative regions, but in many low-resolution polygonal

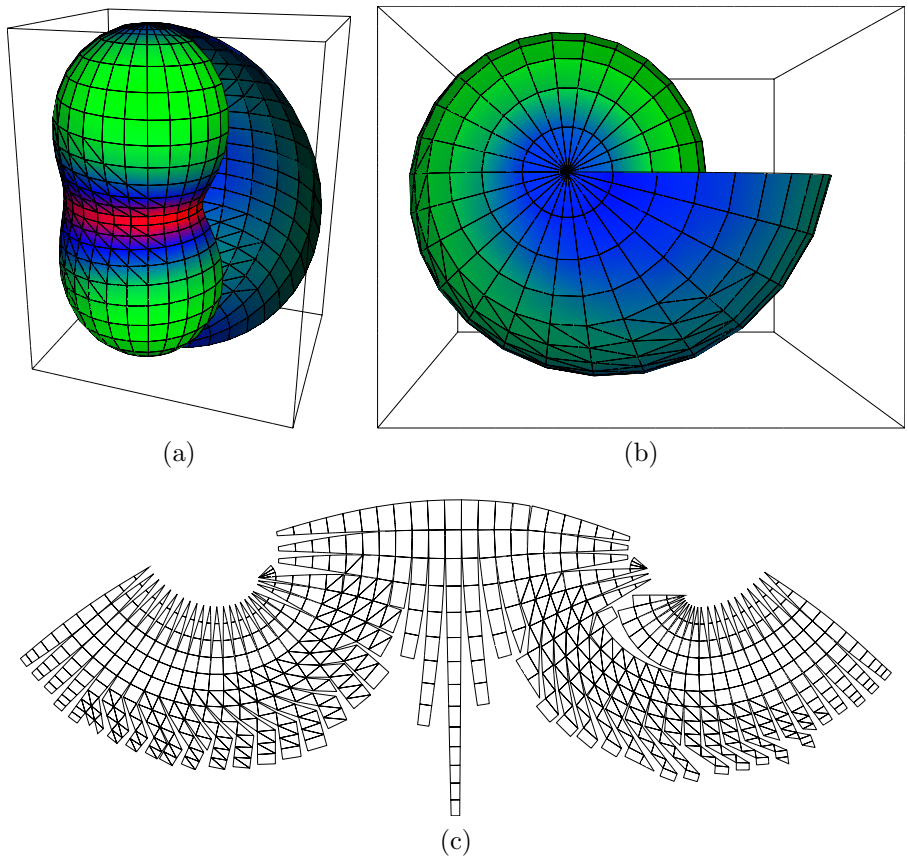


Figure 4.8: A seashell curl, colored by curvature and unfolded by the Curvature Ordering Unfolder (a) Perspective view (b) View from below on the Y axis (c) Unfolded

meshes this is not the case.

4.6.3 Tracer Curvature Ordering Unfolder

The *Tracer Curvature Ordering Unfolder* (TCOU) (Algorithm A.17) is a variant on the Curvature Ordering Unfolder designed to trace across the gradient of curvature flow. The TCOU traces connected strips of faces which cut as perpendicular as possible to the vector of greatest change of angle deficit. This design is much more well-suited to regions of negative curvature than the COU.

The Tracer was designed during early work on the *coolinoid* (Chapter 5) and tends to generate long, spiraling strips of unfolded faces. It suffers from placing too strong an emphasis on continuity of the trace, making it ill-suited to general surfaces.

The COU glues faces across edges which follow the diminishing gradient of curvature, and the COU does well in positive-curvature regions. The TCOU glues faces across edges which run perpendicular to the gradient of curvature and does well in negative-curvature regions. Perhaps a hybrid of the two approaches

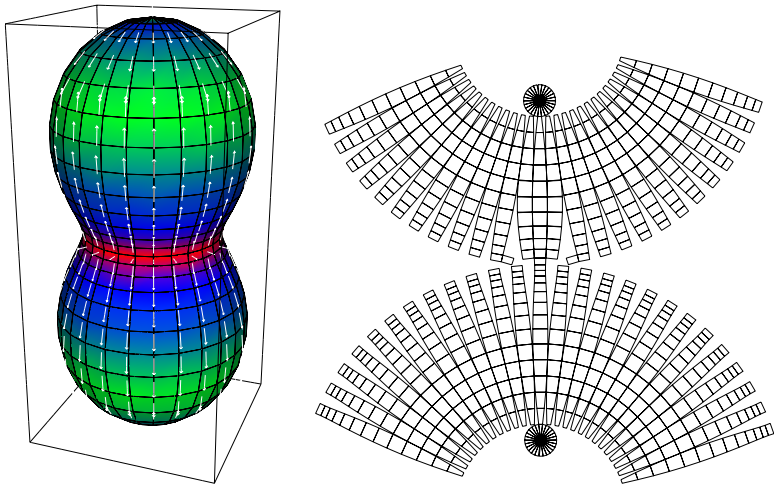
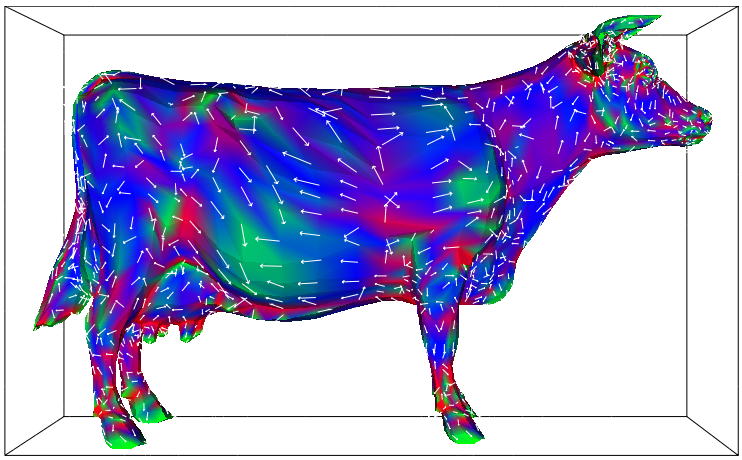


Figure 4.9: A peanut, colored by curvature and unfolded by the Curvature Ordering Unfolder. Note how the Curvature Ordering Unfolder has prioritized the two positive halves, linking them only with a minimal transit across the negative band.



(a)

Figure 4.10: The polyhedral model of the cow, shaded with curvature and flow. Note the negative red shading at each of the major joints and where the ears and horns meet the head, compared with the positive green shading of the flanks.

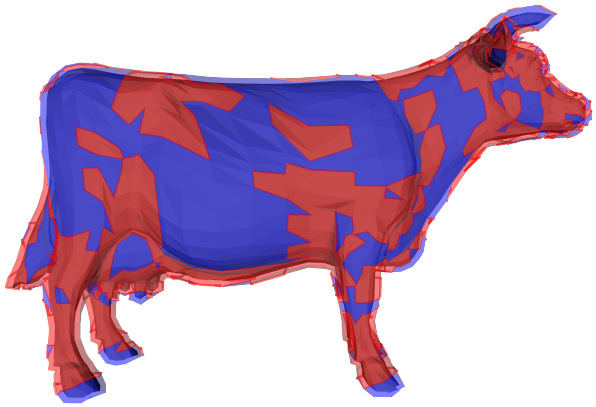


Figure 4.11: The polyhedral model of the cow separated into *regions of convexity* after 1^{st}-order smoothing of the angle deficit field.

would work well.

4.6.4 Region Unfolder

The *Region Unfolder* (Algorithm A.18) expands on the Curvature Ordering Unfolder by extending it to support multiple parallel rooted unfoldings. At its core, it is built on the dual assumption that (a) anything convex can be unfolded and (b) multiple separate unfoldings can be joined together without clashing. The author recognizes freely that both of these are only suppositions–especially the second–but nonetheless this algorithm has shown good potential on some of the simpler non-convex models, such as the peanut (Figure 4.12) and bean (Figure 4.13.)

Here is a brief overview of the sequence of operation of the Region Unfolder:

1. The surface is separated into distinct regions by positive (R_i^+) and negative (R_i^-) curvature.

2. The shortest paths ($S_{i\to j}$) on the surface between each positive region are recorded. Note that these paths must necessarily include faces of negative curvature.

3. Each positive region is independently unfolded to an isolated net, using a recursive call to another unfolding algorithm (the current implementation uses a variant on the Least Height Unfolder.)

4. One positive region R_0^+ is arbitrarily designated as the 'root' of the unfolding.

5. The faces on the paths $S_{0\to i}$ from R_0^+ to each of the other positive regions are progressively glued to the unfolded net of R_0^+ until they reach R_i^+. The unfolded net of R_i^+ is then added to the unfolding, until all positive regions have been joined into a single unified unfolding U.

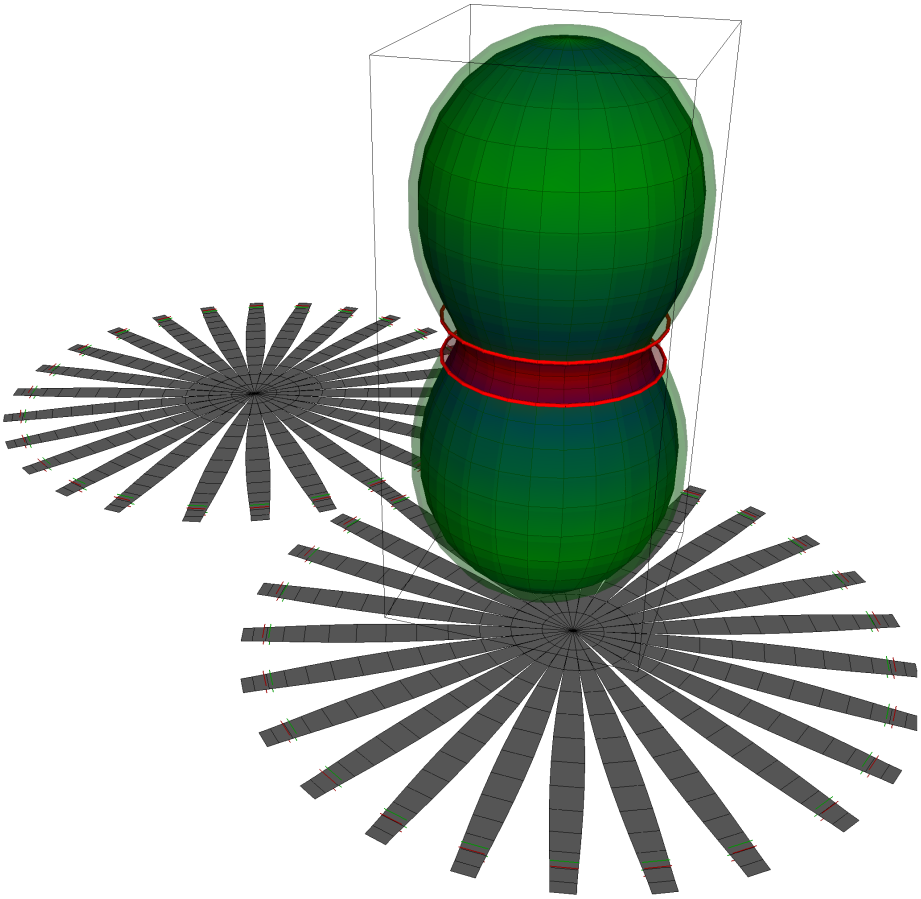


Figure 4.12: The peanut model unfolded by the Region Unfolder. Note the distinctly separated positive regions, each bordered by the slimmer negative triangles. The star-like quality of the upper and lower unfoldings is a side-effect of the border-avoiding unfolding subroutine used for the positive regions.

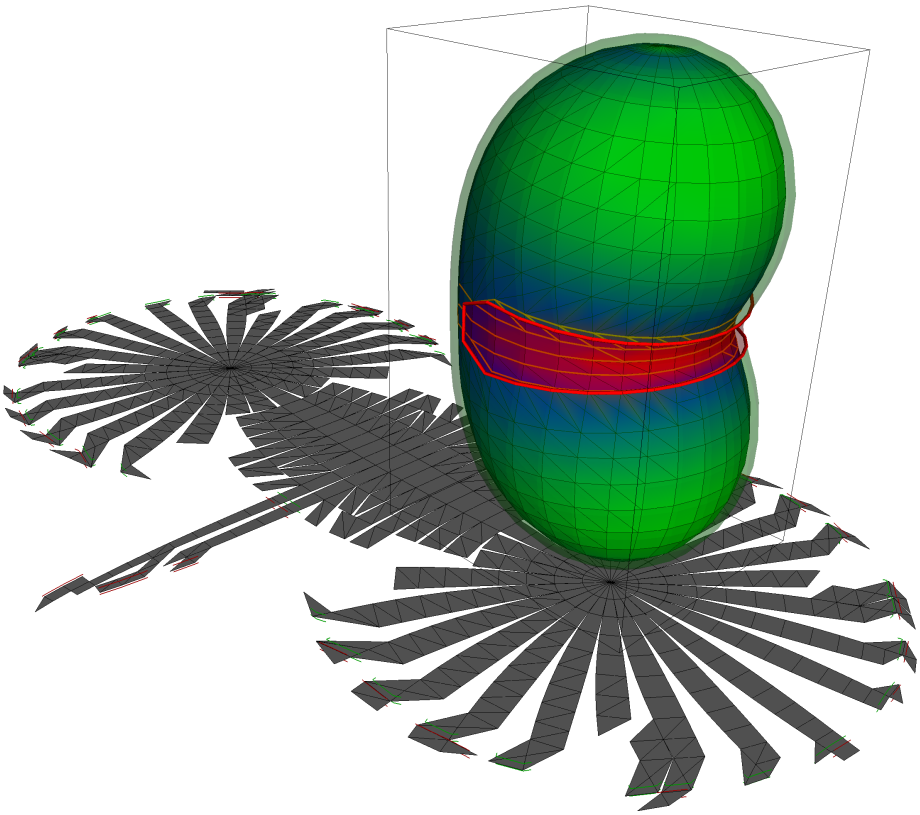


Figure 4.13: The bean model, unfolded by the Region Unfolder. At the jagged tips of each branch of the unfolded net are the dividing lines of red and green which separate positive region from negative. The negative region has been spread by the advancing wavefront across the entire perimeter of the positive region.

6. The border set of undeveloped faces around U then becomes the advancing wavefront of a second unfolding process, adding the negative regions onto the outer perimeter of the positive.

Clearly there are many assumptions in this sketch of the Region Unfolder, and terms left ill-defined:

'Curvature of a face' This is an inaccurate, but apt, description. An individual face of a polyhedron is flat and has zero curvature at every internal point. Instead what is meant by 'curvature of a face' is the average values of the angle deficits of the vertices of the polygon, as originally designed for the first-generation Curvature Ordering Unfolder.

Positive Region In the course of development and testing of the Region Unfolder algorithm, several different definitions of a positive region have been explored. A *positive region* R^+ is:

- a connected set of faces all of which have average curvature greater than or equal to zero. OR,
- a connected set of faces each of which shares an edge with its neighbor with at least one vertex having positive angle deficit. OR,
- a connected set of faces each of which shares an edge with its neighbor with both vertices having positive angle deficit. OR,
- a connected set of faces which form the largest possible convex cap containing the majority of those faces.

In short, a 'positive region' is multiply defined. The author has experimented with each of the above, with mixed results. Interestingly, the best results have come from the last option: finding the largest possible convex cap which contains the most faces of the polyhedron. For insight into why this should be so, consider the curve of a sea-shell spiral (an example is shown in Figure 4.8.) On such a curled surface, faces with positive average curvature may lie deep within the convex hull of the surface, which lends less support to the initial assumption that any positive region on a non-convex surface could unfold without overlap.

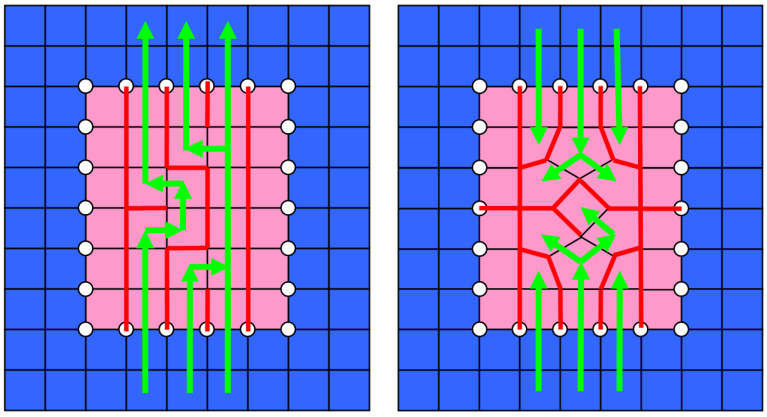
Negative Region A *negative region* R^- is easily defined: each connected set of faces on the polyhedron which do not belong to any positive region (however positive regions are determined) is a negative region.

Shortest Path Again, several different definitions for this term have been considered. A *shortest path* between R_i^+ and R_j^+ is:

- the list of connected faces from R_i^+ to R_j^+ which has the least number of elements. OR,
- the list of connected faces from R_i^+ to R_j^+ which has the least total center-to-center distance of adjacent faces on the polygon. OR,
- the list of connected faces from R_i^+ to R_j^+ which has the least total center-to-center distance of adjacent faces in the unfolded plane. OR,
- the list of connected faces from R_i^+ to R_j^+ which minimizes some as yet-to-be-determined energy function (HP04; NGH04)–perhaps related to curvature?

The current implementation uses the first option. The author has experimented with the second and third. The fourth remains an extremely intriguing avenue of future research. In (NGH04) Ni et al. give an accessible introduction to the application of Morse theory to discrete meshes. Experiments with the other forms of shortest path have not been discouraging, but it is appealing to think that Morse theory might offer better results.

There are two more areas of uncertainty in the design and implementation of the Region Unfolder: the questions of *how*, exactly, to unfold the positive regions and the negative wavefront. In the current implementation, the positive regions are each unfolded by a 'border-avoiding unfolder', a modified version of the Least Height Unfolder which replaces the Y axis with the axis perpendicular to the border of the region. Negative regions are unfolded without heuristic, by gluing



(a) (b)

Figure 4.14: Negative region R^- (pink) is surrounded by positive region R^+ (blue). Red edges indicate the cut-graph; green arrows show the unfolding net. (a) Every vertex in R^- must be cut and no vertex in R^- can be a leaf-node in the graph. But in this figure, the cut-graph has branched, forcing three leaf vertices; these guarantee that the unfolding will self-intersect. (b) Here R^- is being cut apart by a gluing which marches inwards from the border. Encountering an extraordinary face, the net splits and creates an illegal leaf-node on the cut-graph.

each face in the front to the nearest unfolded face and advancing the front until all negative faces have been unfolded (or overlapped.) Future experimentation has room for significant improvement to these subroutines. (Figures 4.12, 4.13.)

The Region Unfolder, in short, is still 'under development', and it remains a promising line of research.

Undevelopable regions of negative curvature

The Region Unfolder depends on its ability to connect positive regions by unfolding across negative regions. It operates on the (not unreasonable) assumption that each of positive regions will be completely unfoldable, but whether the negative regions can link the positives together is another matter, often impossible to prove. A few ground rules can be laid out, however.

Recall that a cut-graph may never have a leaf at a vertex of negative angle deficit, for this would immediately introduce a 1-local overlap. Now consider some negative region R^-, bounded on all sides by a positive region R^+. Clearly, any branch of the cut-graph which enters R^- must leave it again, as it cannot stop amongst negative vertices. Furthermore it cannot branch unless there is another edge out for the branch to exit through. Figure 4.14(a) shows an example of a false branch, where the edges of the cut-graph enter the negative region but branch before exit, forcing the net to follow an odd U-shaped detour through the region. This illegal branch leaves three leaf-nodes stranded within the negative region. This demonstrates that if the number of vertices on the boundary of R^- is k then there can be at most $k - 1$ branch points on the

cut-graph within R^-.

Consider Figure 4.14(b), in which R^- is to be aggressively cut apart, with a cut edge crossing from each vertex in R^+ to each vertex inside R^- across every vertex on the boundary. This is equivalent to a gluing in which all of the faces on the innermost border of R^+ have been separated and spread out across the plane and now the gluing is moving inwards, contracting into R^-. If in the course of the wavefront's advance a situation is encountered where there is one more face than the number of advancing fronts, a leaf node is created in the cut-graph.

This second example demonstrates how a negative region within a positive region could still be completely undevelopable. It seems reasonable to hypothesize that this case might be detected in software, allowing a well-built algorithm to declare success impossible without an exhaustive search.

4.7 Evolutionary Unfolders

The algorithms discussed in previous sections have been, in a sense, 'single-generation' algorithms: that is to say, they treat the surface as it is initially presented, independent of any feedback which might modify the heuristics which choose the next edge to cut or next face to glue. This last section is devoted to the techniques which modify the problem as they solve it: methods which alter the geometry of the polyhedron, or their own heuristics, as they work. Because each of these methods refines its behavior over time, they are grouped together as 'evolutionary' unfolders.

4.7.1 Pattern-matching and simplification

The following is an idea which will not be addressed in detail in this dissertation, but it is interesting to consider: one might match a dense surface with many faces to a known solution for a similar surface with fewer faces. This would depend on being able to map cut-graphs from one polyhedron to another but there is some evidence that this should be feasible[8]. For example, consider polyhedra which are subdivision surfaces: it seems reasonable to conjecture that the unfolding of a low-resolution version of the surface would form a basis of sorts for the unfolding of the same model in high resolution.

Outside of subdivision surfaces, this approach may be untenable. Without knowledge *a priori* of a 'simpler' surface it would likely be necessary to match both topological and geometric features against a dictionary of known unfoldings, particularly at cusps of curvature and along the boundaries of positive and negative regions. This would not be trivial to implement: matching on such a broad scope is a prohibitively difficult task.

4.7.2 Alpha-Beta Unfolder

Section 3.3 details the alpha-beta rules, which use an evolving ordering to dynamically sort the vertices of the surface as the cut-path is discovered.

[8] Personal communications with Joseph O'Rourke, 2008.

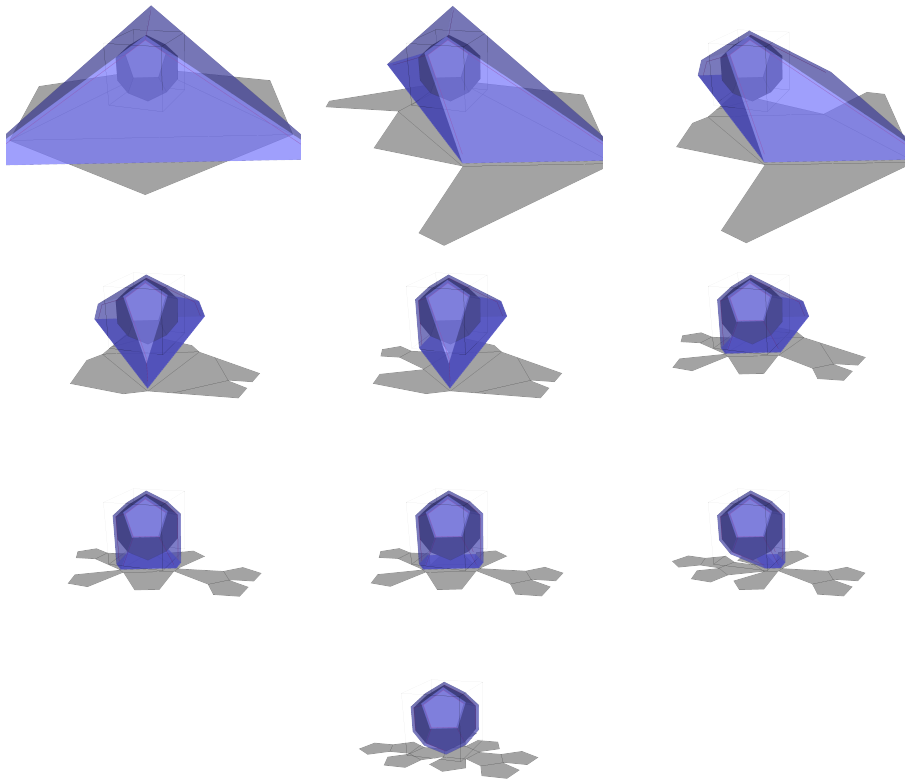


Figure 4.15: The progressive truncation of a dodecahedron

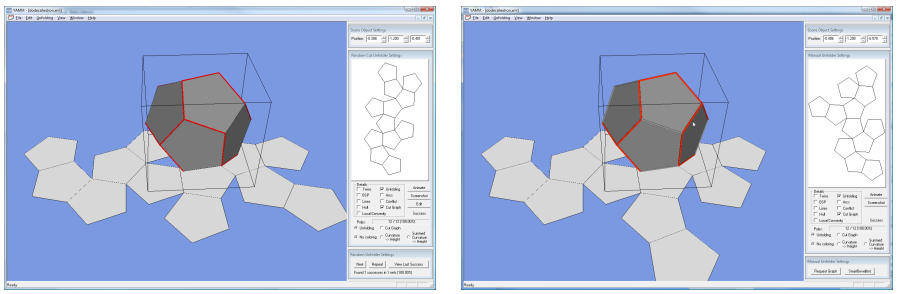
4.7.3 Collision Repair Unfolder

The collision repair algorithm, which iteratively transforms a 'failed' net until it is collision-free, is described in Section 3.4.

4.7.4 Truncation Unfolder

A more viable form of evolutionary unfolder is the progressive refinement of faces from an initially-simple surface. This is the key idea behind Joseph O'Rourke's and the author's *vertex truncation* paper (BO07). The proof of vertex truncation is discussed in detail in Section 2.4 and will not be repeated here. Briefly, the key theorem of (BO07) is that if a given cut-graph has the open sector property then the surface can be truncated at any leaf-node of the cut-graph to refine to a polyhedron with one more face and two more vertices. The new polyhedron will be known to be unfoldable.

The author has implemented a *Truncation Unfolder* (Figure 4.15; Algorithm A.19) which expresses the ideas of the truncation argument and, to a limited degree, extends them. When writing the vertex truncation paper the goal was to prove that certain classes of convex polyhedra are always unfoldable, but the truncation method can be applied with reasonable success to broader classes of convex surfaces which could not be proved recursively unfoldable with the method used in the proof. For example, surfaces which cannot themselves



(a) (b)

Figure 4.16: (a) The Random Unfolder, a testing unfolder in the YAMM software, shown applied to a dodecahedron. (b) The Manual Unfolder, also used in testing; the user has changed a cut edge.

be spanned by a binary tree can still be achieved through a series of truncations which preserve the empty sector property. Figure 4.15 shows the progressive truncation of a large tetrahedron until it is a dodecahedron, with all unfolding knowledge intact.

The Truncation Unfolder is, by its very nature, only suitable for unfolding convex polyhedra. With a more advanced polygon clipping routine it would, theoretically, be possible to adapt the code to handle nonconvex surfaces. To do so, however, would prove significantly more challenging.

4.8 Other Unfolders

4.8.1 Random Cut Unfolder

For testing purposes, the *Random Cut Unfolder* (Figure 4.16(a)) proved invaluable for Monte-Carlo style sampling. However, building a truly randomized unfolder proved to be more challenging than the author had first anticipated. Early versions of the algorithm would always start from a fixed root face (the lowest polygon in the polyhedron) because, when iterating visually through a large number of random unfoldings, a common reference frame is essential. This had the unexpected result of prejudicing the random series: the root face tended to be placed roughly centrally to the unfolding. While every possible unfolding *would* eventually be generated by this method, they were not all being produced with equal probability.

Subsequent versions of the Random Unfolder tried various clever ways to choose a cut-graph at random but it proved difficult to ensure truly random distribution without having to build in expensive loop-removal tests.

The final version of the Random Unfolder returns to its roots: it achieves a truly random series of unfoldings by selecting a root face at random each time it runs and then building a random gluing of faces around that briefly-elected root face. This guarantees (and has been tested) to produce every possible unfolding of a surface with uniform probability.

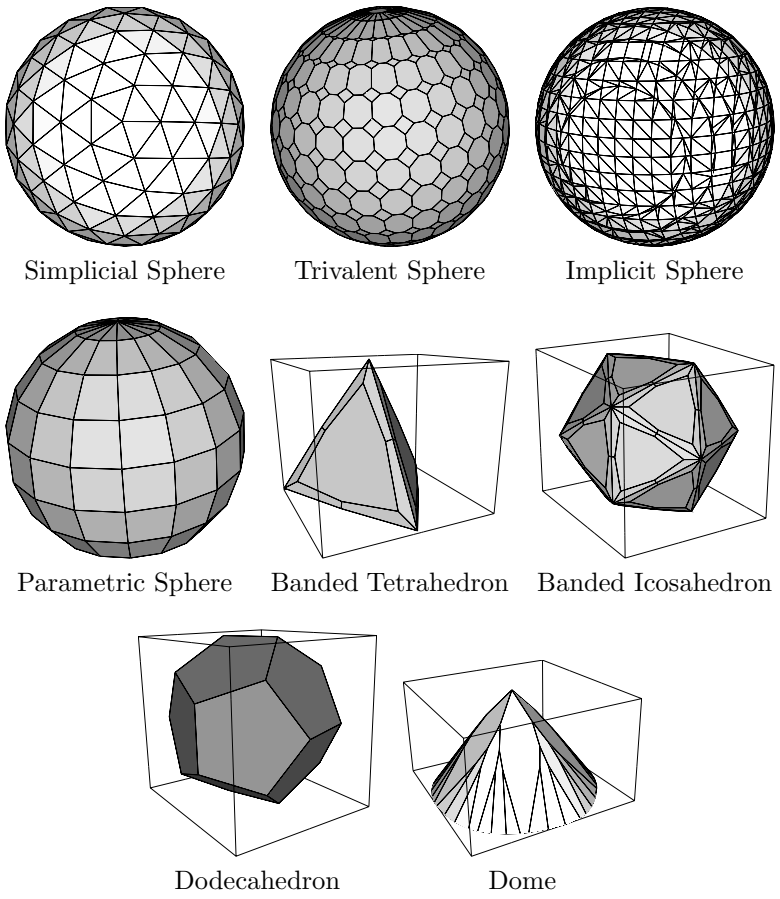


Figure 4.17: Convex Test Surfaces

4.8.2 Manual Unfolder

No dictionary of unfolders would be complete without this humble testing tool: the *Manual Unfolder* (Figure 4.16(b)) which allows a user to graphically edit the cut-graph on the 3D model of the polyhedron and watch the unfolding change 'live'. It has been invaluable in discussion and for finding counterexamples during the author's work.

4.9 Comparison of Results

How do the progressive unfolders compare against each other?

The algorithms were tested on a suite of polyhedra chosen for feature variety and difficulty of unfolding (Figures 4.17 and 4.18.) The non-convex models were deliberately chosen of sufficient complexity that no algorithm would be likely to unfold one correctly 'by accident'. The test models used are listed in Table 4.4.

The algorithms were also exhaustively tested against roughly 200,000 randomly-generated simplicial convex polyhedra, ranging from 4 faces to 300 faces.

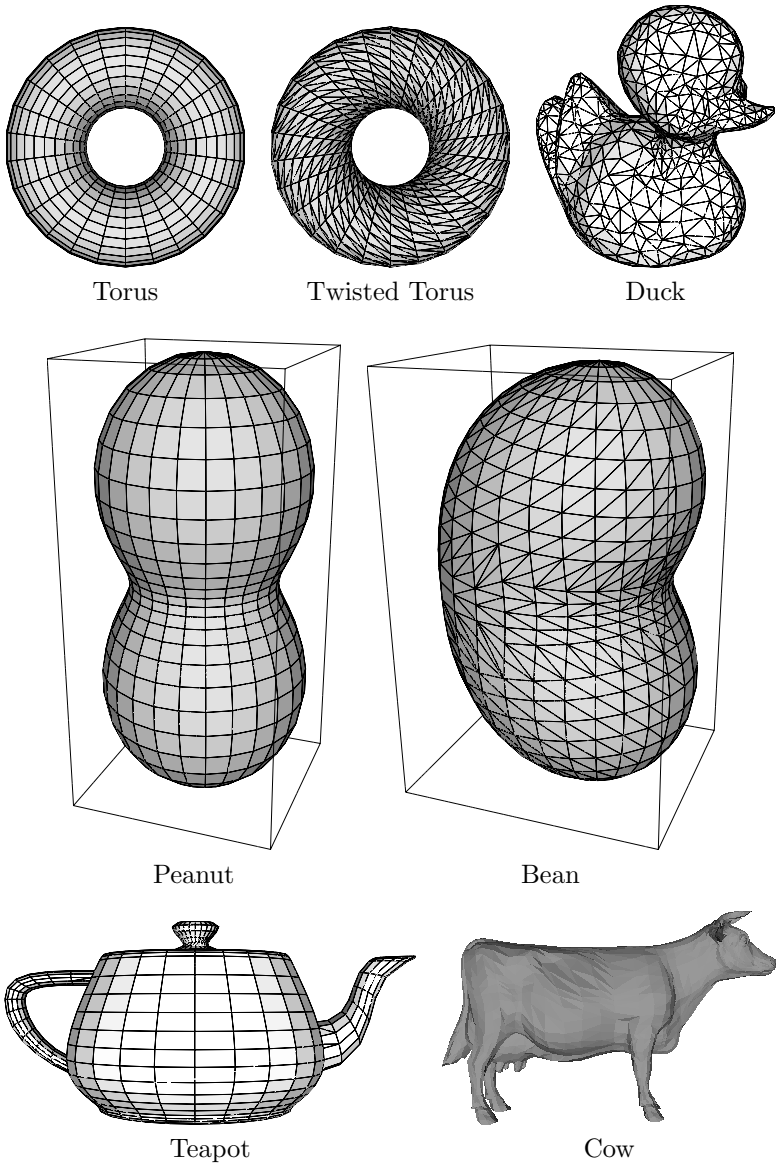


Figure 4.18: Non-Convex Test Surfaces

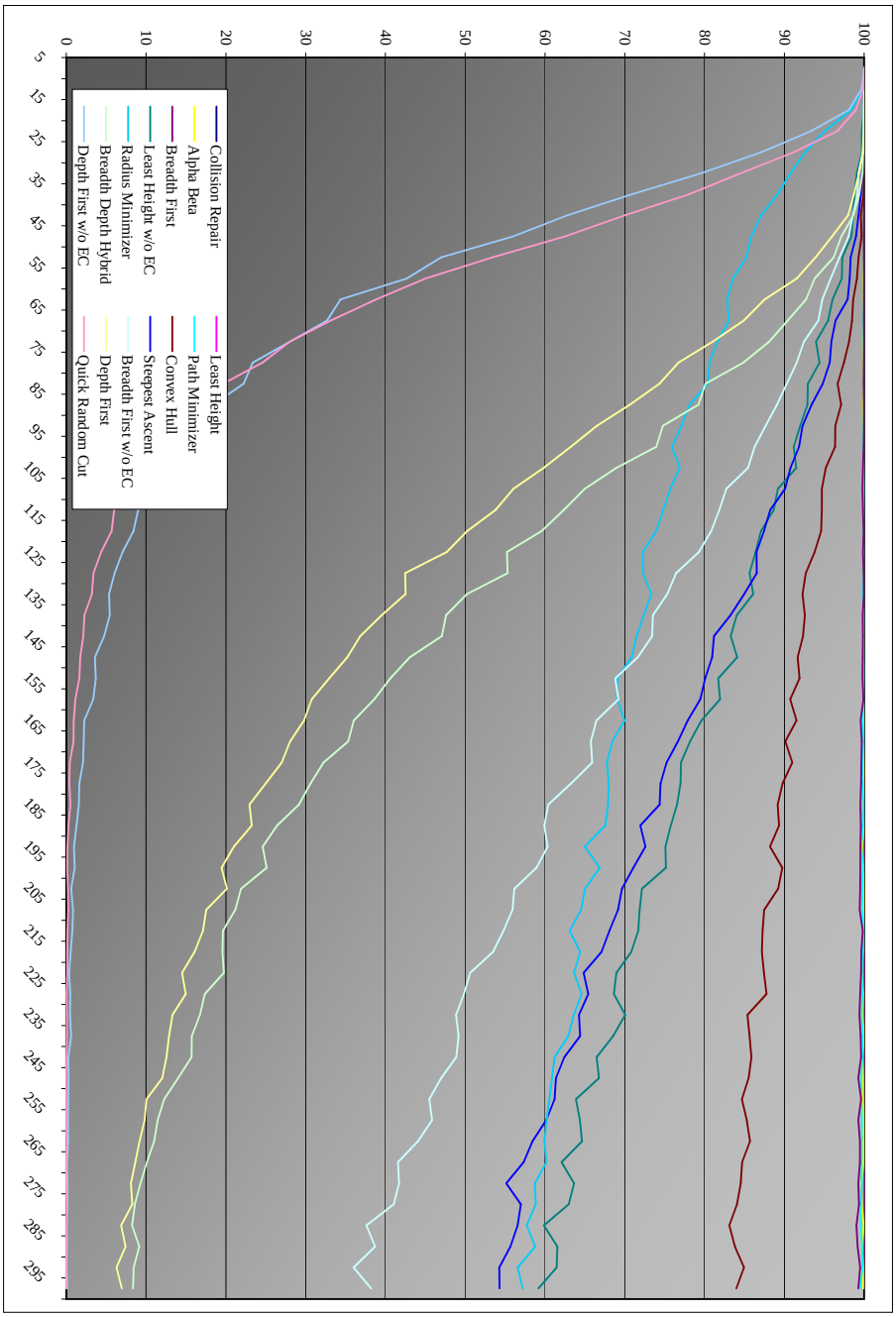


Figure 4.19: Percentage of nets generated without overlap, per unfolder, as the number of faces per random convex polyhedron ranged from 4 to 300. Roughly 200,000 such tests were performed.

Convex:		**Non-convex:**	
Dodecahedron	12 faces	Torus	676 faces
Banded Tetrahedron	28 faces	Peanut	676 faces
Dome	41 faces	Bean	1089 faces
Banded icosahedron	140 faces	Utah Teapot	1269 faces
Parametric sphere	144 faces	Twisted torus	1352 faces
Simplicial sphere	288 faces	Duck	1998 faces
Trivalent sphere	652 faces	Cow	5804 faces
Implicit sphere	2312 faces		

Table 4.4: Test models

Unfolder	**Score**	Peanut	Torus (Twisted)	Torus	Bean	Duck	Teapot	Cow
Curvature	*89.69*	**100**	**100**	**100**	**98.9**	**90.44**	**70.3**	**68.19**
Region	*84.62*	100	100	100	100	83.28	54.28	54.79
Least Height	*84.15*	100	83.95	93.49	100	84.98	63.4	63.23
Breadth First	*77.95*	100	74.78	86.39	95.87	72.72	80.25	35.63
Tracer	*73.47*	74.7	100	53.25	87.14	76.78	66.02	56.41
Avg Success		*80*	*60*	*40*	*40*	*0*	*0*	*0*

Table 4.5: Non-convex tests: percentage of faces per model unfolded without overlap

The Breadth-first, Depth-first, and Least Height Unfolders were also tested with their implicit error correction disabled, for comparison. The difference in performance between the two versions of each of these three unfolders is quite noticeable.

Of the algorithms investigated, three stand out above the rest: the Alpha-Beta Unfolder (Section 3.3), Collision Repair (Section 3.4) and the Least Height Unfolder (Section 4.5.3.) Each of these, Collision Repair above all, is a candidate for the hypothetical 'perfect unfolder', the algorithm which by its existence and its inability to fail proves that all convex polyhedra are unfoldable. However, it must be understood that statistical evidence is not a proof.

Tables 4.5 and 4.6 are organized as follows: a column of Unfolders is contrasted against a row of unfolding tests. Where row and column meet, the table shows the percentage of faces which were *not* overlapped in that Unfolder's unfolding of that test. The Unfolders are sorted top-to-bottom in descending order of 'Score', which shows the average rate of overlap-free faces per Unfolder. (This average is weighted by the unfolding, not by the number of faces per unfolding.) The test models are also sorted left-to-right in descending order of average 'Success', i.e., the percentage of tests which were able to unfold the test without overlap.

From Table 4.6 it is clear that the Alpha-Beta, Collision Repair, Breadth First and Path Minimizer Unfolders performed best on the chosen convex test models. Amongst the non-convex Unfolders, Curvature Ordering outperformed all others.

Figure 4.19 on page 141, shows the relative performance of each of the progressive (convex-surface-oriented) Unfolders on large sets of randomly-generated

Unfolder	**Score**	Dodeca-hedron	Dome	Parametric Sphere	Simplicial Sphere	Trivalent Sphere	Banded Tetrahedron	Implicit Sphere	Banded Icosahedron
Alpha Beta	*100.00*	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Breadth First	*100.00*	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Collision Repair	*100.00*	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Path Minimizer	*100.00*	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Radius Minimizer	*99.19*	100	100	100	98.75	98.16	100	96.63	100
Breadth Depth Hybrid	*99.08*	100	100	98.61	99.38	99.23	100	96.84	98.57
Depth First	*98.78*	100	100	100	99.38	98.16	100	96.24	96.43
Unravel	*97.79*	100	100	100	97.81	91.72	100	97.1	95.71
Convex Hull	*97.32*	100	100	100	100	100	89.29	100	89.29
Least Height	*96.61*	100	100	100	100	100	78.57	100	94.29
Breadth First w/o EC	*96.00*	100	100	100	100	100	78.57	99.39	90
Least Height w/o EC	*95.16*	100	100	100	100	100	78.57	99.87	82.86
Depth First w/o EC	*94.77*	100	100	98.61	92.81	96.17	89.29	90.53	90.71
Steepest Ascent	*93.69*	100	100	100	100	100	78.57	98.05	72.86
Star	*93.28*	100	80.49	100	100	100	78.57	100	87.14
Random Cut	*85.01*	100	60.98	92.36	92.81	76.53	78.57	88.15	90.71
Average Success		*100.00*	*87.50*	*81.25*	*62.50*	*62.50*	*50.00*	*43.75*	*31.25*

Table 4.6: Convex tests: percentage of faces per model unfolded without overlap

simplicial convex polyhedra. This chart is interesting for several reasons. It clearly places Alpha-Beta, Least Height and Collision Repair at the head of the class for random polyhedra; all three functions follow straight lines at the top of the chart while others die away as polygon face count increases. The yellow curve tracks the Random Unfolder, and (comfortingly) the odds of the Random Unfolder unfolding a polyhedron without overlap drop off along the same curve as that found by Schevon in (O'R00). Interestingly, the Depth-first Unfolder actually *under-performs* the Random Unfolder for polyhedra of up to roughly 60 faces.

4.10 Conclusions

This chapter has introduced a number of implementations of unfolding algorithms, ranging from brute-force and simple heuristic solutions to curvature-based and evolutionary techniques. A comparison of the performance of the various Unfolders has been presented. The reader has been introduced to concepts such as implicit error correction and the classification of Unfolders by their decision basis.

The Region Unfolder has been introduced and some positive results for this curvature-aware unfolder of nonconvex surfaces have been outlined. However, the Region Unfolder is clearly still a very young line of research and much more work will be required before it is of practical use.

One excellent direction for future research would be to combine Collision Repair with the Region Unfolder. It would be necessary to adapt Collision Repair to support 1-local overlap.

Testing has shown that the Collision Repair, Alpha-Beta, Breadth-first, Least Height and Path Minimizer Unfolders outperform all others, with Collision Repair leading the pack. If none of these methods is actually perfect then they are certainly on the right path.

Chapter 5

Unfolding the Coolinoid

5.1 Introduction

It is known that there are convex polyhedra which are unfoldable but which can be unfolded to overlap, such as Fukuda's tetrahedron (Figure 1.5, page 13.) It is also known that there are nonconvex polyhedra which can never be edge-unfolded, such as the Witch's Hat assembly (Figure 1.6, page 14.) But there has been little research into the class of surfaces which lies between these two negative results: the developable nonconvex polyhedra.

This chapter reproduces the paper *A Developable Surface of Uniformly Negative Internal Angle Deficit* presented by the author at Mathematics of Surfaces XII, 2007 (Ben07). It discusses the *coolinoids*, a class of nonconvex polyhedra with negative curvature at every vertex which is partially developable.

This chapter extends the original paper with a proof that the coolinoid must be unfolded with the spiral unfolder (Section 5.4), which was left as a supposition in the original work.

5.2 Cycles in the Cut-Graph, Angle Deficit and Developability

By definition, the unfolding tree cannot possess cycles and must be connected. The cut graph may possess cycles; the circumstances in which this is permitted may be bounded explicitly:

Lemma 5.1 *The presence of a loop in the cut graph requires that there be at least one handle in the topology of the polyhedron.*

Proof. A loop in the cut-graph of the surface forms a Jordan curve on the surface which separates any locally $\Re 2$ surface into exactly two discrete components[1]. If the source mesh is without handle (locally $\Re 2$) then the unfolding tree will be broken into two parts. Therefore either the surface must have a

[1]From (Gri76), Appendix A.2, p.95. Griffiths does not prove this theorem directly, instead citing Newman's *Elements of the Topology of Plane Sets of Points*, Cambridge University Press, London (1954), p. 137.

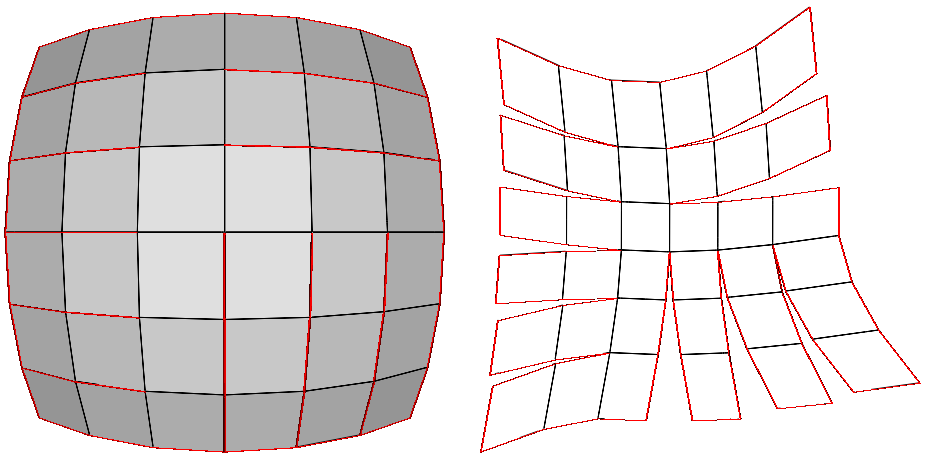


Figure 5.1: Positive curvature: edge cuts terminate at inner vertices

handle which connects the faces within the loop to the faces outside the loop; or, the cut-graph cannot loop. □

Corollary 5.2 *The formation of a loop in taking the union of the cut graph with the set of all boundary edges of the polyhedron requires that there be at least one handle in the topology of the source mesh.*

Proof. Boundary edges, like cut edges, have no dual in the unfolding tree. Loops in the (potentially disjoint) graph of boundary edges are permissible but, as in Lemma 5.1, new loops may not be introduced in taking the union of the cut graph and the boundary graph on a simply-connected surface. To do so would separate the surface into two or more disjoint parts as above. □

Recall from (Pol03) that at every spherical vertex there need be only one incident cut edge but that at every hyperbolic vertex at least two incident edges must be cut. This gives the following lemma:

Lemma 5.3 *No simply-connected surface of uniformly negative internal angle deficit is developable.*

Proof. Suppose that there exists a simply-connected surface of uniformly negative internal angle deficit which is developable. To generate the unfolding, a cut graph must exist. This graph cannot loop, as the surface is simply connected; likewise, it may contain at most one vertex which lies on a boundary edge of the surface . If the graph does not loop then it must have at least two leaf nodes, where the edges of the cut graph originate and terminate, but all vertices have negative curvature. Therefore no such surface may exist. □

Corollary 5.4 *The number of branches in the cut-tree which develops a mesh of uniformly negative internal angle deficit cannot exceed $B-2$, where B is the number of distinct boundary curves on the mesh.*

Proof. The number of leaf nodes of a tree is two plus the sum of the valence minus one of each branch node in the tree. Thus each branch raises the

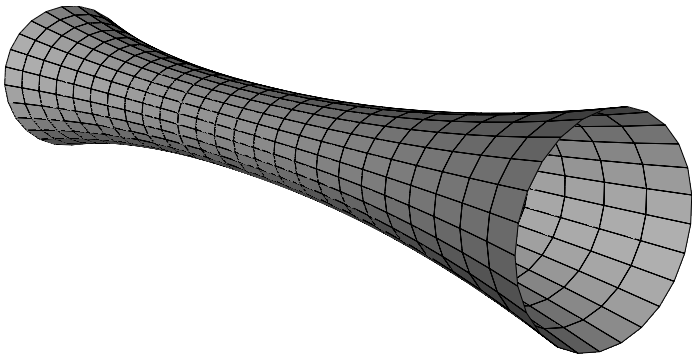


Figure 5.2: Coolinoid

required number of boundary curves by one. As above, there must be at most one leaf node per boundary curve and no leaf node may not fall on a boundary curve. □

Corollary 5.5 *The (topologically) simplest developable surface of uniformly negative internal angle deficit has at least two boundary curves.*

Proof. It would be impossible for a graph to have -1 branch points. □

5.3 A Developable Surface of Uniformly Negative Internal Angle Deficit

A *coolinoid*[2] is the surface of rotation obtained by rotating a polynomial $f(t) = t^k + 1$ about the t axis. To obtain a discrete representation, the following parametric description is used:

```
for (u:=0 to 1 step 1/du)
  for (v:=-1 to 1 step 2/dv)
  {
    x = cos(2\pi * u) * (pow(v,k)+1);
    y = v * h;
    z = sin(2\pi * u) * (pow(v,k)+1);
  }
```

where $k \geq 1$ and $h > 0$. The model shown in Figure 5.2 was generated from ($h = 10.0$, $k = 2.2$, $du = 25$, $dv = 25$). Informally, h may be thought of as the 'stretch' of the model, k may be seen as the 'curviness' of the model and du and dv are the 'resolution' of the mesh.

Many surfaces whose internal angle deficit is negative at every vertex are not developable, and several examples of such surfaces are shown in Figure 5.4. However, *there exist configurations of the coolinoid which are developable.* The model shown in Figure 5.2 unfolds into the spiraling net shown in Figure 5.3. The net has no self-intersection and is completely planar.

This demonstrates that

[2]so named for its resemblance to the cooling tower of a power plant

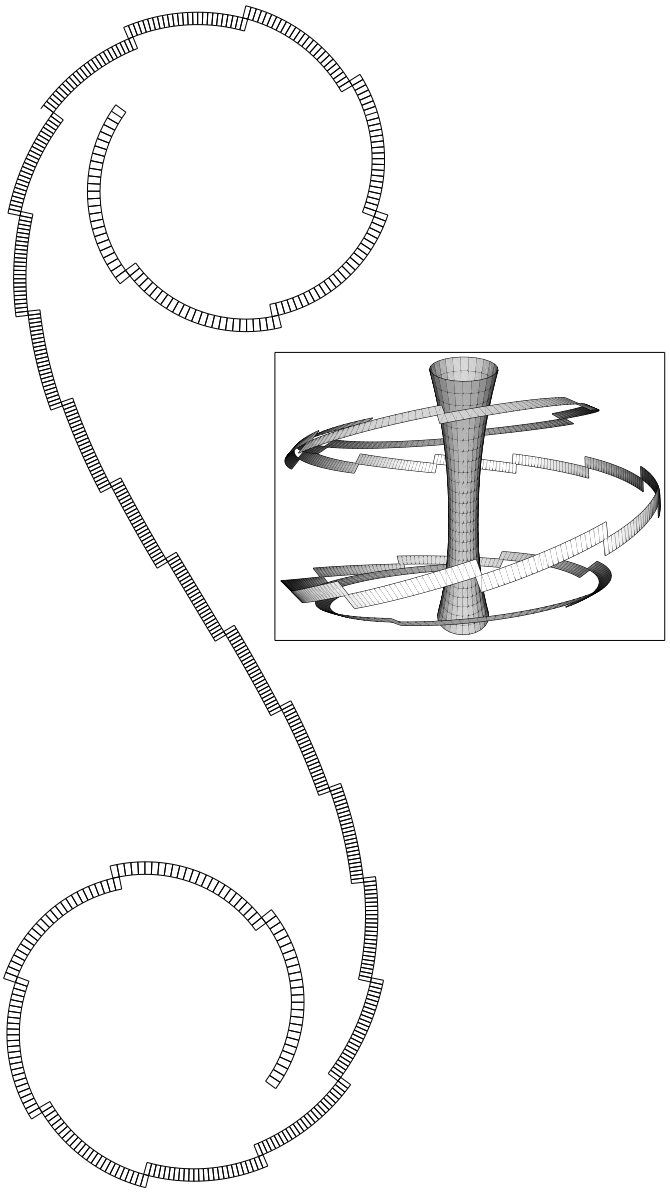


Figure 5.3: Spiral unfolding of a coolinoid ($h = 10.0$, $k = 2.2$, $du = 25$, $dv = 25$.) Inset: The spiral unfolding in progress.

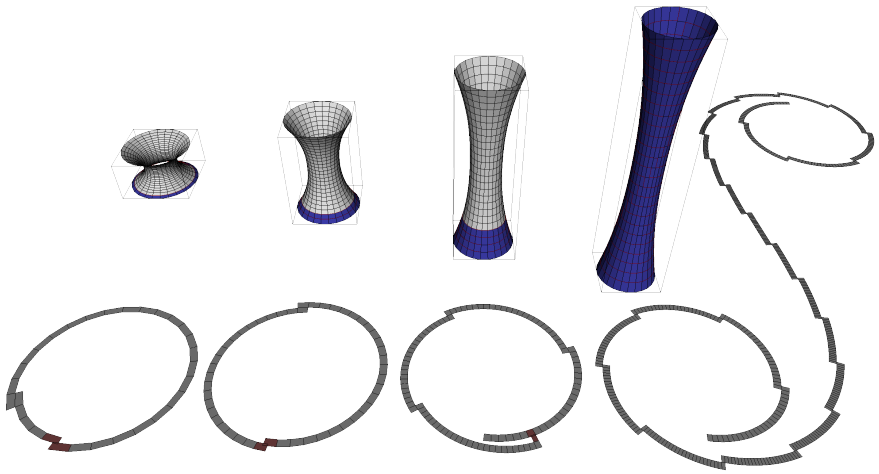


Figure 5.4: Developability is dependent on construction. Here k is fixed at 2.25, du and dv are fixed at 25, and $h = 1.2$, $h = 3.2$, $h = 6.2$ and $h = 9.2$. Only $h = 9.2$ is developable (blue faces have been unfolded.)

Theorem 5.6 *There exists a connected surface of uniformly negative internal angle deficit which is developable.*

Proof. By example (Figure 5.3.)

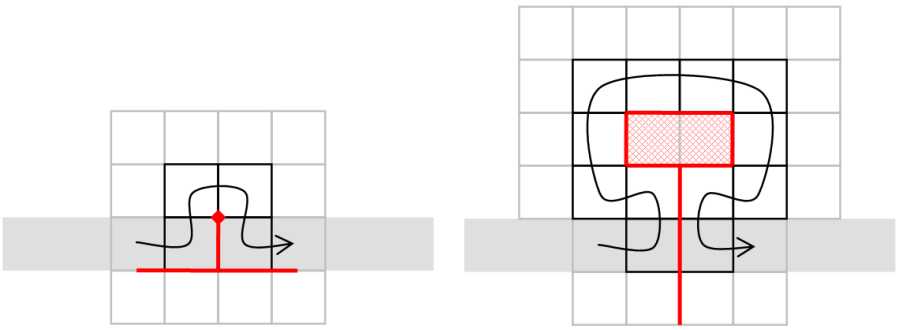
Note: Numerical verification that the model shown in Figure 5.3 has no self-intersection may be obtained by verifying that Equations 1 and 2, below, have no common solution for $h = 10.0$, $k = 2.2$, $du = 25$, $dv = 25$. □

5.4 Spiral Unfolding

Corollary 5.4 allowed restriction of development to linear stripping heuristics: as there can be no inner branch, the unfolding must be a Hamiltonian path. Furthermore it must be a Hamiltonian path which never generates a leaf-node on the graph of its border, a very limiting constraint. The only Unfolder which meets these criteria is the Unravel Unfolder (Section 4.5.3), which peels the model in a spiraling strip of consecutive faces ordered by adjacency in a counterclockwise traversal about the positive Y axis and then by ascending Y value.

Lemma 5.7 *Any coolinoid which can be unfolded and which is not exactly a cylinder, can only be unwrapped by a strip which spirals around the axis of the coolinoid.*

Proof. The solution cannot branch, therefore only stripping solutions are considered. For discussion purposes, assume that the strip is oriented, beginning on the lower boundary of the coolinoid and proceeding upwards and to the right; all arguments hold from the top down or from right to left. The strip must travel from row one to row dv.



(a) (b)

Figure 5.5: *Rule 1*: The strip can never pass through two adjacent faces in opposite directions.

Rule 1: The strip can never pass through two adjacent faces in opposite directions.

Proof. To do so would either result in a leaf-node of the cut-graph (Figure 5.5(a)) or faces trapped in a loop (Figure 5.5(b)).

—

Rules 2: The strip must travel the length of each row before proceeding to the next one.

Proof. The strip enters the grid of the faces of the coolinoid at some entry face a on row one of the grid. From there it proceeds to the right along k faces before turning upwards at face b (Figure 5.6(a).)

Assume that $k < du$. Then the strip leaves row one at point b and must later return at some point c (Figure 5.6(b).) By Rule 1, c cannot be adjacent to a or b: it is moving upwards as it enters a and leaves b but the strip travels downwards as it enters c. There are then four possibilities:

1. Figure 5.6(c): The path returns to row one and forms a loop, trapping faces between b and c. The path cannot branch therefore these faces would remain inaccessible and never be unfolded. If there were no faces inside the loop then the loop would necessarily contain (at least one) leaf-node of the cut-graph. Therefore the path cannot reach c from the left and then turn to the right.

2. Figure 5.6(d): The path returns to row one and curls inwards, trapping itself between faces b and c. The path cannot branch or cross itself and so the tip of the path would be trapped inside the curl, unable to reach the top row of the coolinoid. Therefore the path cannot reach c from the left and then turn to the left.

3. Figure 5.6(e): The path loops around the cylinder of the coolinoid before reaching c, then traps faces within a loop as it travels toward b. As in case 1, the trapped faces could never be unfolded. Therefore the path cannot reach c from the right and then turn to the right.

4. Figure 5.6(f): The path loops around the cylinder of the coolinoid before reaching c, then curls to the right. As in case 2, the head of the strip

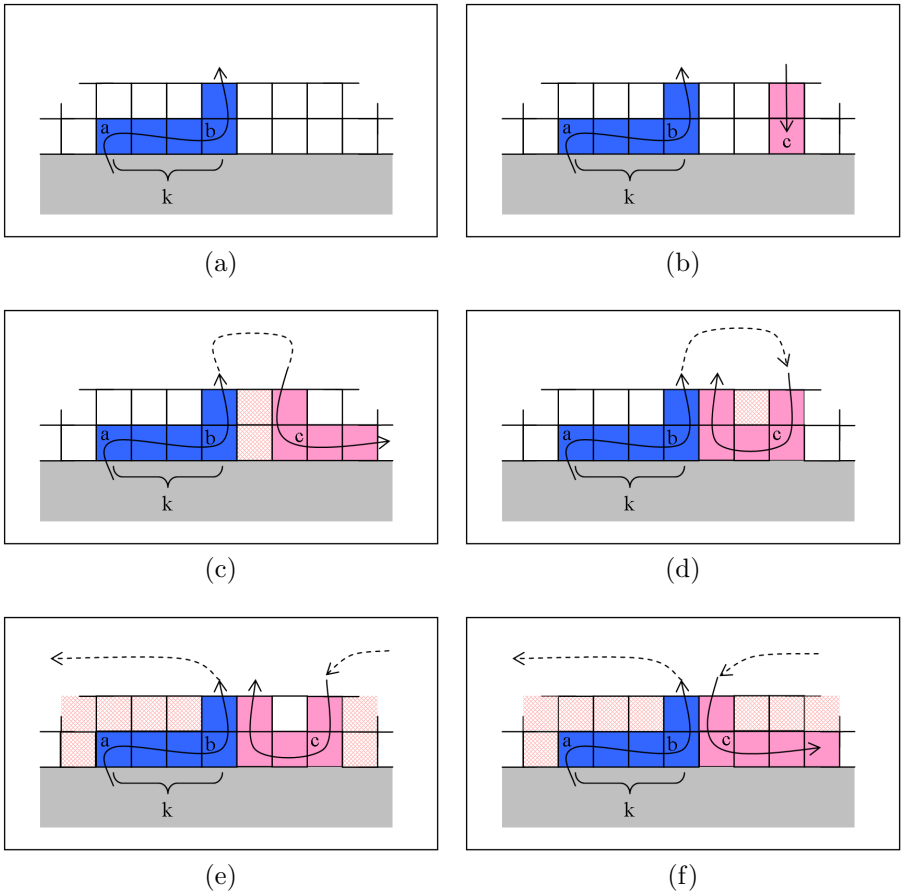


Figure 5.6: *Rule 2*: The strip must travel the length of each row before proceeding to the next one.

would be trapped within its own loop, unable to reach the top row of the coolinoid. Therefore the path cannot reach c from the right and then turn to the left.

With all four possibilities invalid, the assumption $k < du$ must be discarded. Therefore $k = du$ and the strip fills all of each row before moving on to the next row.

—

Rule 3: The strip cannot zig-zag.

Proof. By rule 2 the strip must complete all of row one before advancing to row two. Assume that it travels left-to-right to fill row one, then at the last cell of row one it steps up to row two. At this first cell on row two it cannot turn back to travel to the left: doing so would violate rule 1. Therefore it must continue onwards to the right, wrapping around the cylinder.

—

Thus by rules 1, 2 and 3 the strip must spiral around the coolinoid. No other unfolding is possible. □

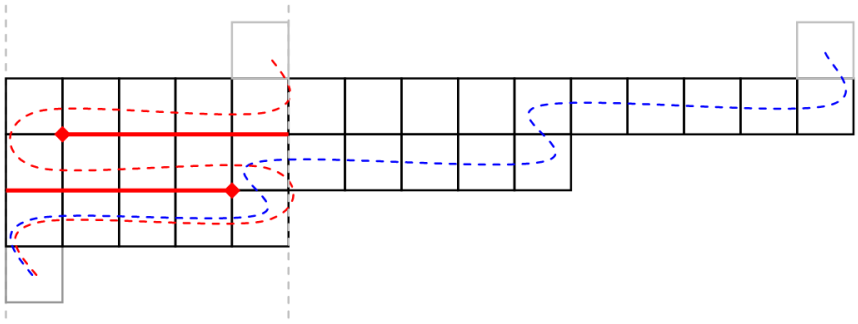


Figure 5.7: *Rule 3*: The strip cannot zig-zag.

5.5 A Black-Box Solution for Determining the Unfoldability of a Coolinoid

The observation that developability of the coolinoid varied as a function of h, k, du, dv spurred the development of a 'black-box' solution which could determine *a priori* whether a given coolinoid model could be unfolded. Note that for general non-convex polyhedra, there exists no known method faster than exhaustive sampling for determining unfoldability.

Unwrapping a single row of faces from a coolinoid generates a curved arch of trapezoids which does not self-intersect; the flattened strip could never complete a full circle unless it already was one (which is the special case of the inner core of a torus, and undevelopable.) Each subsequent row, unrolled into a strip attached to the last face of the previous strip, will have less curve than the previous row until the unwrapping crosses the vertical midplane, after which the curved arch of each row will wrap symmetrically in the opposite direction. This creates a smooth spiral which unravels to a straight line and then curls back into another spiral.

However, not all coolinoids can be spirally unfolded. The shared edge of the last face of a row i and the first face of row $i+1$ is the edge which lies *inside* the curl of the spiral; this moves each row inwards, toward the center of the spiral, by the height of the row. For many coolinoids this inwards step introduces a subsequent conflict between two rows, as shown in Figure 5.4(a-c).

5.5.1 A functional expression of the unfolding of a coolinoid

The curve of the lower and upper borders of the spiral unfoldings of the faces of a coolinoid can be expressed as functions of a single linear parameter.

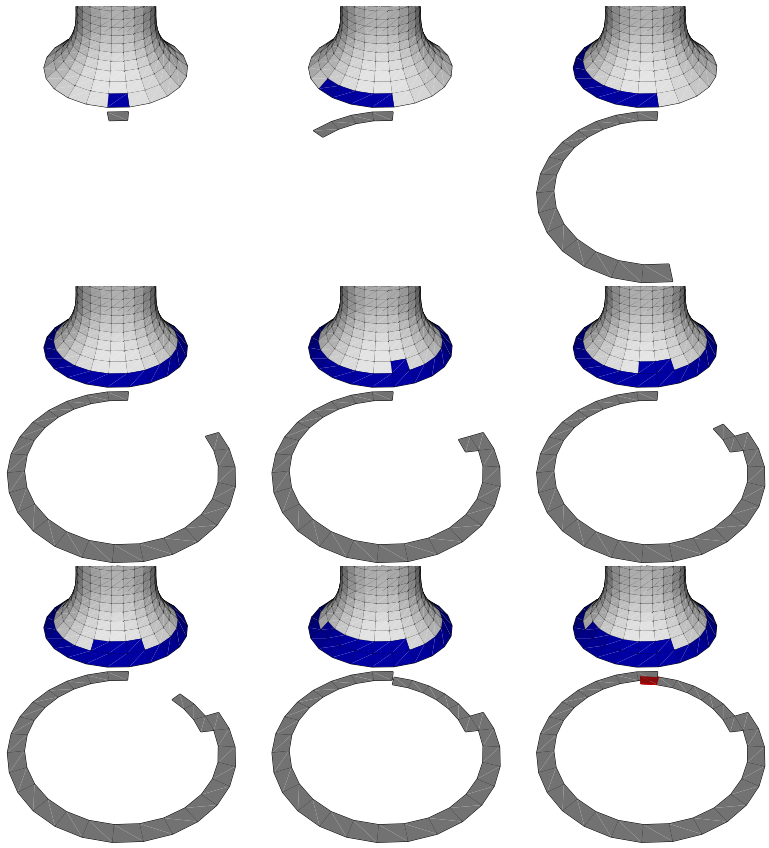


Figure 5.8: Spiral unfolding with conflict in the second row

Taking h, k, du and dv as constants, define:

$$\begin{aligned}
P(u,v) &= \left[\left(\cos\frac{2\pi u}{du}\right)\left(|\frac{2v}{dv-1}|^k+1\right), \frac{2vh}{dv-1}, \left(\sin\frac{2\pi u}{du}\right)\left(|\frac{2v}{dv-1}|^k+1\right)\right] \\
Up(v) &= P(0,v+1)-P(0,v) \\
Over(v) &= P(1,v)-P(0,v) \\
\alpha(v) &= cos^{-1}\left(\frac{Up(v)}{\|Up(v)\|}\cdot\frac{Over(v)}{\|Over(v)\|}\right) \\
\beta(v) &= \pi-2\alpha(v)
\end{aligned}$$

where

- $P(u,v)$ is the Coolinoid function. u, v range from 0 to $du-1$ and $dv-1$, respectively.
- $Up(v)$ is the step from the 'bottom' of row v to the 'top' of row v.
- $Over(v)$ is the step from one vertex at the bottom of row v to the next vertex in the row, ordering the vertices in a clockwise direction up the positive Y axis.

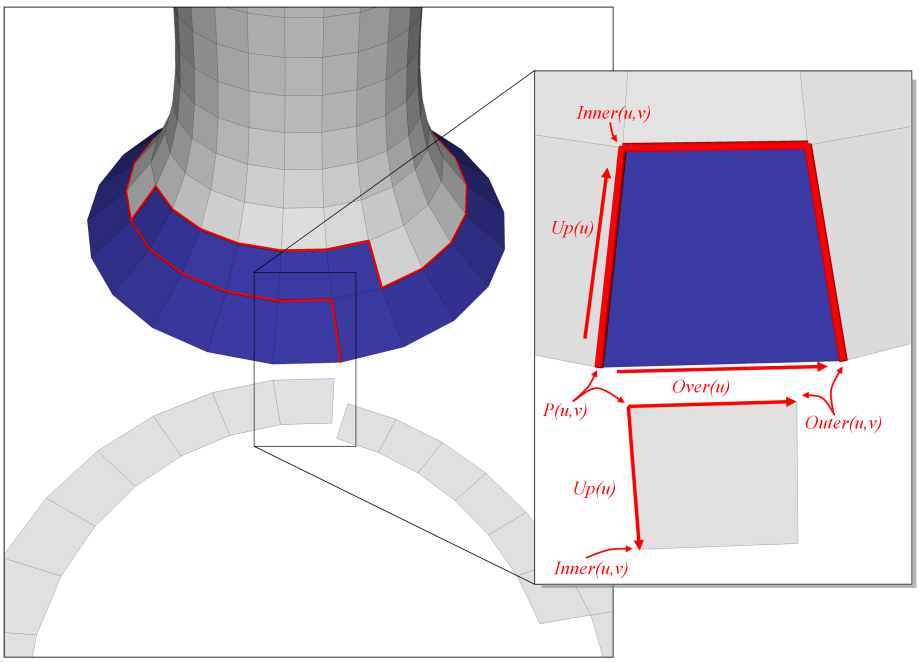


Figure 5.9: Terminology used in the functional expression of a coolinoid unfolding

- $\alpha(v)$ is the acute inner angle of the trapezoidal faces of row v.
- $\beta(v)$ is the angle by which the unwrapping of row v will 'curl' in the plane at each face.

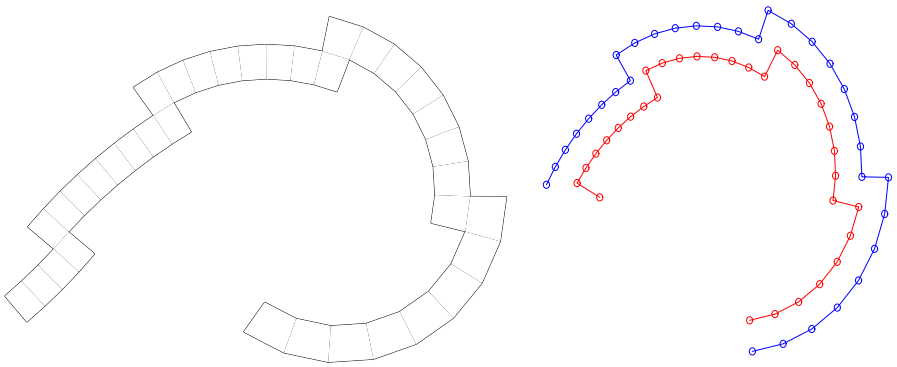
These symbols are shown in place in Figure 5.9.

Caution: this use of α and β is not to be confused with the unrelated overloading of these symbols in the alpha-beta rules of Chapter 3.

The following functions are then defined:

$$
\begin{aligned}
Turn(v) &= \sum_{lvl=0}^{v} (du-1) * \beta\,[lvl] \\
Hop(v,\theta) &= \|Up(v)\|\,[cos(\theta), sin(\theta)] \\
Skip(v,\theta) &= \|Over(v)\|\,[cos(\theta), sin(\theta)] \\
Jump(u,v) &= Hop(v-1, (Turn(v-1) + \alpha(v-1) + \beta(v-1))) + \\
&\quad \sum_{i=1}^{u} Skip(v, Turn(v-1) + i * \beta(v)) \\
Outer(u,v) &= Jump(u,v) + \sum_{lvl=0}^{v-1} Jump(du-1, lvl) \\
Inner(u,v) &= Outer(u,v) + Hop(v, Turn(v-1) + \alpha(v) + (u+1) * \beta(v)) + \\
&\quad \lfloor u/(du-1) \rfloor * (Hop(v+1, Turn(v) + \alpha(v+1)) - Skip(v+1, Turn(v)))
\end{aligned}
$$

such that



(a) (b)

Figure 5.10: (a) An unfolding (b) The same unfolding evaluated in Mathematica^{TM}

- $Turn(v)$ is the total curl introduced by the unwrapping of row v.
- $Hop(v, \theta)$ gives the vector $Up()$ rotated by θ.
- $Skip(v, \theta)$ gives the vector $Over()$ rotated by θ.
- $Jump(u, v)$ gives the offset of the unwrapping of the first u faces of row v.
- $Outer(u, v)$ gives the position of the lower right corner of the unwrapping of the first $v - 1$ rows and the first u faces of the $v^t h$ row.
- $Inner(u, v)$ gives the position of the upper right corner of the same faces as $Outer(u, v)$. The construction of $\lfloor \frac{u}{du-1} \rfloor$ is designed to shift the function back one face, allowing the first face of each row to share its lower edge with the last face of the row preceding.

The functions $Outer()$ and $Inner()$ are now defined over the range $\{u, v\} \in \{[0 \ldots du - 1], [0 \ldots dv - 1]\}$ with u and v both held integer.

The unraveling unfolding evaluates these functions in a linear progression. Encapsulating this linear progression as

$$\begin{aligned} OuterRing(t) &= Outer(\lfloor t \rfloor mod(du), \lfloor t/du \rfloor) \\ InnerRing(t) &= Inner(\lfloor t \rfloor mod(du), \lfloor t/du \rfloor) \end{aligned}$$

yields two univariate equations whose solution(s), if they exist, are the loci of intersection of the outer and inner border of the unfolding (Figure 5.10.) Testing for the unfoldability of a coolinoid is now reduced to solving for the intersection of these two equations.

5.6 Developability of the Coolinoid

As mesh resolution increases, the odds of the mesh being unfoldable decrease (Figure 5.11) echoing similar results gathered by J O'Rourke in Figure 2 of

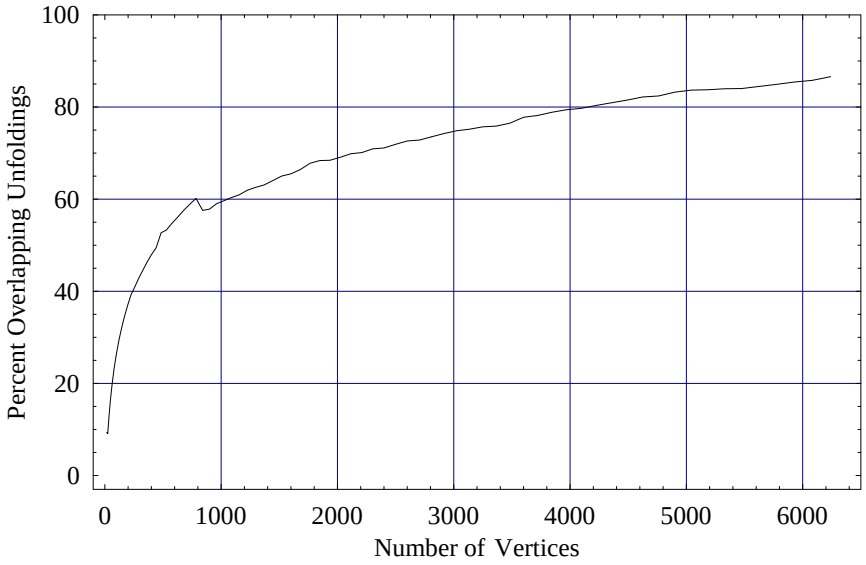


Figure 5.11: Overlap vs Dimension

(O'R00). The data shown in Figure 5.11 was gathered by taking the average developability over the range $\{h, k\} \in \{[0.5 \ldots 14.75], [0 \ldots 10]\}$ for each integer value of dim in the range $[3 \ldots 78]$, setting $du = dv = dim$.

Evaluating the developability of coolinoids continuously over the domain given above, an implicit surface is generated (Figure 5.12.) The surface shows extremely intriguing behavior. Predictably, it displays interleaved shelving effects in some areas, highlighting regions of $\{h, k, dim\}$ space where parity (odd/even) of the mesh dimension has an effect on the developability; this is most readily visible where $k < 1$. Such liminal regions are common in such surfaces. More interestingly, an evolving wave pattern–hinting at fractal behavior–begins to appear in the isosurface as dimension increases. The wave hints at a much more complex isosurface than might be expected. Other ranges of symmetric and asymmetric behavior appear throughout the surface; further study is warranted.

In figure 5.12 the isosurface is shown looking from up the positive h axis. The positive k axis travels left-to-right and dim travels from the image's bottom to top. Note that the dim axis is integer, creating a voxel-like shelving effect (the 75 shelves in Figure 5.12) along the vertical axis.

Figure 5.13(a) shows the isosurface from the side, looking down the positive k axis toward the origin.

Figure 5.13(b) shows a detail of the lower h, k values. Note the interleaved parity-sensitive structure close to the origin in the $k \leq 1$ region, followed by a deep trough of undevelopability in the range $1 \leq k \leq 2$.

Figure 5.13(c) shows an overhead view, looking down on the surface from the h axis. Note the wave pattern along the topmost border of the isosurface.

Figure 5.13(d) shows the isosurface from above, looking directly down the positive dim axis. The wave in the isosurface is clearly visible.

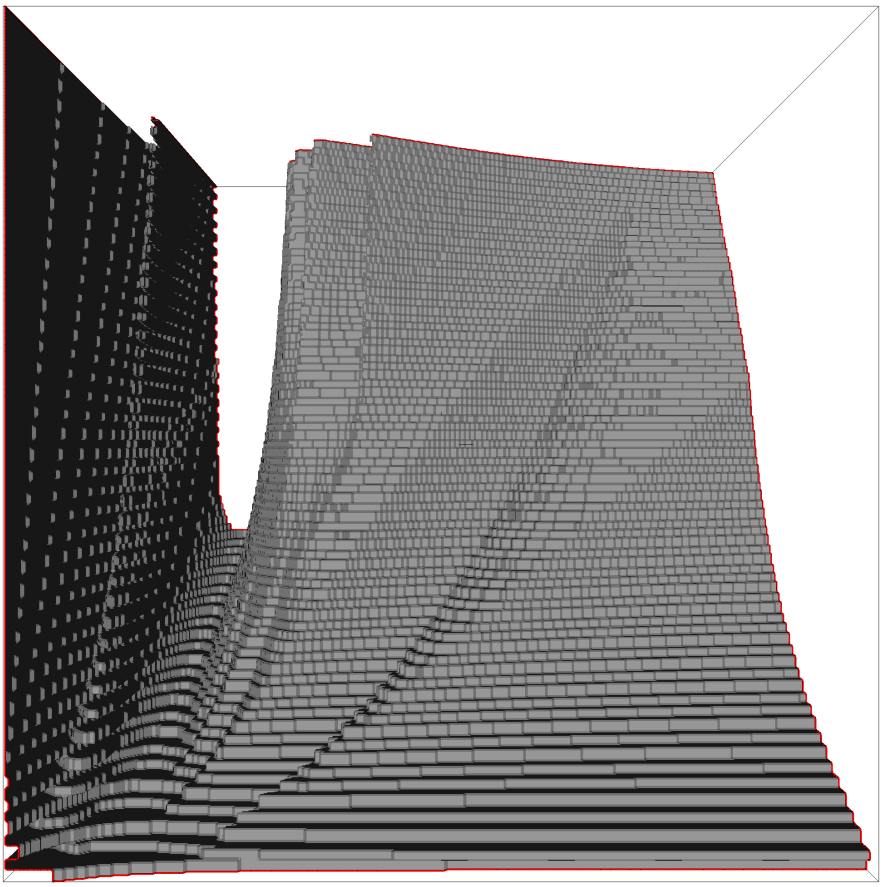
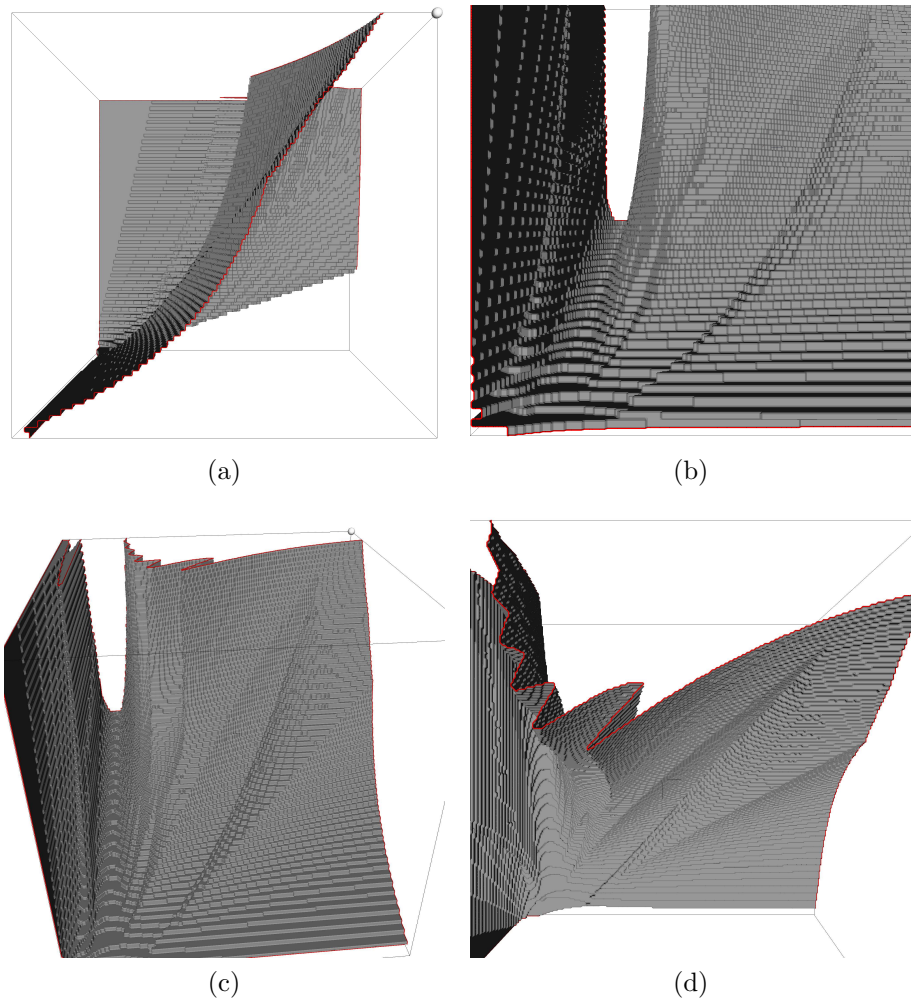


Figure 5.12: Coolinoid unfoldability represented as an implicit surface in $\{h, k, dim\}$



(a) (b)

(c) (d)

Figure 5.13: Coolinoid unfoldability represented as an implicit surface in $\{h, k, dim\}$

5.7 Future Work

The coolinoid is very similar to functions such as $x^2 + y^2 - z^2 = 1$ and the *Catenoid*. Analysis comparable to that performed above would be quite instructive. The techniques demonstrated will apply to any surface which may be unfolded by the spiral unfolder.

The wave function which emerges in the upper ranges ($dim \rightarrow 50+$) of the coolinoid unfoldability isosurface displays fascinating fractal behavior which calls for ongoing investigation. Joseph O'Rourke has suggested[3] that the wave is an artifact of the stepwise nature of the integer *dim* field in conjunction with h, a progression already becoming visible in Figure 5.4 a-d.

The interleaving effects for low-resolution models (Figure 5.13b, lower left) decay as *dim* rises. Does that decay flatten fully, or is it re-expressed at higher resolutions in the much subtler interleaved effects that appear at higher values of h?

In the isosurface shown, it was assumed that $du = dv$. It would be very interesting to decouple these two fields, plotting a four-dimensional isosurface, substituting one of the four axes for time and animating the evolution of the wave.

5.8 Conclusions

It has been shown that simply-connected surfaces of negative interior curvature cannot be unfolded. An example has been given of a developable surface of negative interior curvature with two boundary curves: the *coolinoid*. A black-box solution for determining the unfoldability of any given coolinoid has been found and further analysis of the developability of the coolinoid has yielded startlingly complex and intriguing results.

[3]Personal communications, February 2007

Chapter 6

Conclusions

This dissertation began with two questions, and while neither question has been answered to the author's satisfaction, significant strides have been made in both areas. It is the author's sincere hope that the material which has been presented here will be of use to future researchers.

6.1 Can it be shown that all convex polyhedra can be edge-unfolded?

The quest continues for the proof that all convex polyhedra are unfoldable, or the counterexample of the undevelopable polyhedron. A number of new contributions have been made in both of these directions.

A series of negative results have been presented on unfolding convex polyhedra, including the following:

- Many convex polyhedra have at least one unfolding which contains overlap, but every convex polyhedron seen thus far has at least one overlap-free development;
- Unfolding along the shortest edge-paths from a common point (that is to say, unfolding in the same manner as the general Star Unfolding) will often avoid overlap, but not always;
- Unfolding on an open convex curve will not always avoid overlap

The technique of *cut-tree truncation* has been introduced. Cut-tree truncation allows the description of broad classes of unfoldable convex polyhedra, if they can be derived by degree-3 leaf truncation from a seed polyhedron which has the *open sector property*. Cut-tree truncation has been used to show that the *domes*, a class of convex polyhedra, are unfoldable.

It has been shown that, just as it is possible to show that classes of polyhedra are unfoldable, it is also possible to show that classes of cut-graph will always unfold their targets without overlap, provided that they can be constructed. One such example is the *locally-convex cut-graph*. Several counterexamples have been given of polyhedra which cannot be spanned by a locally-convex cut-graph but the proof holds despite these limitations; it even leads to a second proof of the unfoldability of the domes.

Another such class of algorithmically-driven net and cut-graph is the *Anchorable Convex Hull*, an early foray into unfolding proofs. While the ACH is limited in application to only a few classes of convex polyhedra, specifically those which can be spanned by closed convex curves, it has nonetheless inspired several subsequent directions of research.

In support of the hunt for the undevelopable polyhedron, the technique of polyhedral banding has been formally introduced, analyzed and discussed as a tool in aid of future research. It has been shown that the class of *banded geodesic spheres* are 'provably hard' to unfold, making them excellent candidates for the testing of unfolding algorithms.

On a related note the reader has been introduced to an idea for a proof in the style of Euclid, which seeks to show that there can be no smallest ununfoldable polyhedron (and therefore no ununfoldable polyhedra at all.) Although the first proposed structure for this proof has been rebutted, the concept behind it remains entirely sound.

Two other ideas for proofs have also been proposed:

- The alpha-beta rules seem to hold almost a complete proof, though several barriers do remain. They are also well-supported by testing evidence.

- Collision repair, a second idea which could become a complete proof, lacks theoretical backing; it is extremely well-supported by testing evidence but the precise mechanism of its success is not yet fully understood.

Both of these ideas are strongly deserving of future research.

6.2 How can an arbitrary polyhedron be edge-unfolded?

The reader has been introduced to the Unfolders, a suite of algorithms which can unfold a surface. Different methods of classifying and co-ordinating Unfolders have been explored. The largest differentiation has been between those specific to nonconvex surfaces, those specific to closed convex polyhedra, and those without specialization. Unfolders have also been divided into those which draw on heuristic knowledge of polyhedra, explicitly or implicitly, to build a net or cut-graph, versus those which operate completely independently of the source geometry. They may also be categorized by their decision class–the mechanism by which they choose their next step.

To address the issue of the unfolding of nonconvex polyhedra the Curvature Unfolder and its direct descendant the Region Unfolder have been introduced. In a problem space where failure is known to be possible it is difficult to say that these new algorithms are 'better' than their unspecialized peers; but simple sample models such as the curl, bean and peanut all show that these new methods do contain useful advances. Furthermore the Region Unfolder has also served to introduce and begin to formalize the concept of *negative* and *positive regions*, and the possibility of using knowledge of such regions to quickly negatively determine a surface's unfoldability.

Many of the available Unfolders have been compared side-by-side, and several–the Collision Detection, Alpha-Beta and Least Height Unfolders, above all

others–have shown remarkably positive results. While statistics are no substitute for a robust proof of correctness, there can be no question that these are now, in one researcher's words, "the algorithm to beat."

6.3 Explorations of Interest: The *Coolinoid*

The reader has been introduced to the *coolinoid*, a developable surface of negative interior angle deficit with two boundary curves. The coolinoid is noteworthy because it is topologically the simplest such surface possible; no surface with fewer boundaries and uniformly negative interior curvature can be unfoldable. It is also noteworthy for the subtle patterns in its developability, which can produce fascinatingly fractal results when modeled with an isosurface.

In a new proof, it has been shown that the only possible algorithm for unfolding a coolinoid is a spiral strip.

6.4 Final Thoughts

Throughout this research the author has used a number of models, including scanned surfaces. One surface, in particular, became somewhat iconic for the author, both for its humor value and for its difficult combination of positive and negative curvature. This motivating surface has driven much of the work presented here.

Despite the progress made, a question which was asked three years ago still remains open today: *is* a cow unfoldable?

Appendix A

Unfolding Algorithms

Notes:

Pseudocode is not a typed language and so variable type must be inferred from initialization of a name. Notation in these algorithms distinguishes initialization ($\leftarrow$) from assignment ($=$, $+=$, $\times=$, etc).

The operator '++' is taken from the C programming language. An expression of the form $(++i)$ should be read as, 'i is incremented and the value of i' is returned. An expression of the form $(i++)$ should be read as, 'the value of i is stored in a temporary location, i is incremented, and the temporary is returned.'

The conditional operator '?:' is also borrowed from C. An expression of the form $(A?B:C)$ should be read as, 'if (A) then return B else return C'.

The component operator '.' is borrowed from C++. An expression of the form $A.B$ should be read as 'the B of A'. For example, $f.center$ would be the center of face f; $v.normalized()$ would be the unit-length vector in the direction of vector v.

Lower-case names are used for geometric primitives. For example, f is often used for a face and e for an edge.

Exceptional geometric primitives, names which maintain a constant value throughout the course of an algorithm, are named with capital letters. For example, F_B is used by several algorithms to denote 'the lowest face in the polyhedron, sorting faces lexicographically with the Y axis primary.'

Capital names are used for larger data structures. Common names include:

- T indicates a tree
- G indicates a graph
- P names the input polyhedron
- U indicates an unfolded net

Values in {curly braces} are values in a list. The empty list is $\emptyset$.

Values in <angle brackets> indicate an ordered data structure.

An array is indicated with [square brackets]. For example, $A[i]$ names the i^{th} element in array A.

Addition of edges to trees or graphs is indicated with the '+=' operator and a directed (for trees) or undirected (for graphs) pair. For example, $T+= (A \rightarrow B)$

would add the directed edge AB to tree T; A must be in T. $G+= (A \leftrightarrow B)$ would add the undirected edge AB to graph G; if A and B are not already in G, the edge is still added and the graph is now disconnected.

Subroutines names are written in typewriter font when a subroutine is declared or called. Subroutine names are usually one or more English words with each first letter capitalized. For example, `HeuristicMinimizer`(P, F_B, $Score()$) declares a function `HeuristicMinimizer` which takes three parameters, the third of which is itself a pointer to a function variable. Where a subroutine is used without having been defined, the function of the subroutine is to be inferred from context.

A few common subroutines are:

- `NumEdges`$(f) \rightarrow$ the number of edges in face f
- `GetNeighbor`$(f, i) \rightarrow$ the i^{th} neighbor of face f (index order is assumed to be lexicographic unless stated otherwise.)
- `Copy`$(X) \rightarrow$ a new instance of identical type and value to X

The Total Unfolder

In the Total Unfolder algorithm, W is the tree of all orderings of all unfoldings. Each node $w =< U, L >$ of W consists of an unfolding net U and a list L of available edges to which a new face may be glued. For a given node w in W, for a given edge e in $w.L$, F_e is the face incident to edge e which is not in the unfolding net $w.U$.

Algorithm A.1: The Total Unfolder

1: $F_B \leftarrow$ root face
2: $W \leftarrow$ a tree of a single node, whose value is the pair $< F_B, \{$the edges of $F_B\} >$
3: **for** each $w \in W$ **do**
4: **for** each edge $e \in w.L \| F_e \notin w.U$ **do**
5: $l \leftarrow \{$the new edges of $F_e\}$
6: $W+= (w \rightarrow < w.U + F_e, w.L - e + l >)$
7: **end for**
8: **end for**

The Precomputed Unfolder

The Precomputed Unfolder builds up a sparse array A of unfolding nets. Each net is represented as a graph of faces; faces which share an edge in a graph will share a developed edge in the corresponding unfolded net. Note that all non-$\emptyset$ graphs are guaranteed to have no loops or be disconnected and are therefore actually undirected trees.

Algorithm A.2: The Precomputed Unfolder

1: $L \leftarrow$ an ordered list of the faces of P such that every face in the ordering shares an edge with some previous face in L.
2: $num \leftarrow 1$
3: $max \leftarrow \prod_{f \in L} (\texttt{NumEdges}(f))$
4: $A \leftarrow$ an array of max graphs
5: **for all** $f \in L$ **do**
6: **for** $j = 1$ to $\texttt{NumEdges}(f)$ **do**
7: **for** $k = 0$ to num **do**
8: $A[j \times num + k] = (A[k] \neq \emptyset ? \texttt{Copy}(A[k]) : \emptyset)$
9: **end for**
10: **end for**
11: $num \times= \texttt{NumEdges}(f)$
12: **for** $j = 0$ to $\texttt{NumEdges}(f)$ **do**
13: $f_j \leftarrow \texttt{GetNeighbor}(f, j)$
14: **for** $k = 0$ to num **do**
15: **if** $f \leftrightarrow f_j \notin A[k]$ **then**
16: **if** $f \ldots \leftrightarrow \ldots f_j \notin A[k]$ **then**
17: $A[k] += (f \leftrightarrow f_j)$
18: **else**
19: $A[k] = \emptyset$
20: **end if**
21: **end if**
22: **end for**
23: **end for**
24: **end for**

The Iterative Unfolder

The Iterative Unfolder builds up a list L of graphs, each of which corresponds to a unique unfolded net. It does so by storing what is essential an arbitrary-length, variable-base integer in the array named $INDEX$ and mapping each value of this extended integer to a unique possible unfolded net. The array $INDEX$ therefore has n elements (where n is the number of faces in P).

The subroutine `Inc` increments an extended integer array by one. It does so by cascading incrementation of the integers in $INDEX$, exactly like binary addition, except that where binary addition always wraps a digit at 2, `Inc` wraps the j^{th} digit at the number of faces of the j^{th} face.

The subroutine `MakeUnfolding` converts an extended integer into a graph of face-to-face connectivity, which represents a possible unfolded net. This graph may contain loops and my be disconnected. The graph is assembled by using the value of each 'digit' of the extended integer $INDEX$ as the index of the edge by which to glue the corresponding face to the net.

Subroutine 1: `Inc`$(INDEX)$

```
1: j ← 0
2: INDEX[j]++
3: while INDEX[j] = NumEdges(f_j) do
4:    INDEX[j] = 0
5:    INDEX[++j]++
6: end while
```

Subroutine 2: `MakeUnfolding`$(INDEX)$

```
1: G ← ∅
2: for i = 0 to n do
3:    f ← the i^th face of P
4:    k ← INDEX[i]
5:    G+= f ↔ GetNeighbor(f, k)
6: end for
7: return G
```

Algorithm A.3: The Iterative Unfolder

```
1: F_B ← the lowest face of P
2: L ← ∅
3: INDEX ← an array of n integers, initialized to zero
4: repeat
5:    repeat
6:       Inc(INDEX)
7:       G = MakeUnfolding(INDEX)
8:    until G does not loop and is connected
9:    L+= G
10: until INDEX overflows
```

The Spanned Unfolder

To put it simply, the Spanned Unfolder generates every spanning tree of the faces of P using somebody else's solution. That is to say, algorithms to enumerate spanning trees are an active field of ongoing research; the Spanned Unfolder uses the current best such solution as a 'black box' solution finder, and returns the result as a list L of graphs which map to valid unfolded nets.

Algorithm A.4: The Spanned Unfolder

```
1: G ← the graph of face-to-face connectivity of P
2: L ← {all spanning trees of G}
```

The Family Tree Unfolder

The Family Tree Unfolder chooses an ordering of the faces of P and then, as i goes from 1 to n, computes the complete set of spanning trees of the convex cap containing the first i faces in the ordering. The set of spanning trees is

then mapped to a superset relationship with the set computed for $i-1$. The final output is a disconnected graph G where each edge of G represents a subset relationship.

Note that this is not an Unfolder in the usual sense. Rather it is an algorithm for identifying relationships between generations of unfoldings.

Algorithm A.5: The Family Tree Unfolder

```
1: L ← an ordered list of the faces of P such that every face in the ordering
   shares an edge with some previous face in L.
2: G ← ∅
3: for each f_i ∈ L do
4:    P_i ← P_{i−1} + f_i
5:    G_i ← the graph of face-to-face connectivity of P_i
6:    L_i ← { all spanning trees of G_i}
7:    for each g ∈ L_i do
8:       if ∃g′ ∈ L_{i−1} ‖ g′ ⊂ g then
9:          G+= (g′ → g)
10:      else
11:         G+= g (adds a valence-0 node to G)
12:      end if
13:   end for
14: end for
```

The blind search Unfolders

The Depth-first and Breadth-first Unfolders were first formalized by Wolfram Schlickenrieder in (Sch97). They are extended here with implicit error correction: for each face visited in the chosen expansion of the graph of face-to-face connectivity of P, if adding this face at the current edge would introduce overlap in the unfolding, the face is not added and the corresponding edge in G is pruned.

Note that the blind algorithms as presented here will leave some faces undeveloped if they unfolded to overlap at every attempt. How failure to unfold without intersection is handled is left as an exercise for the implementation. In the YAMM test software, after all conflict-free unfoldings have been made, a second pass over the remaining faces of P unfolds them into their final overlapped positions.

Algorithm A.6: The Breadth-first Unfolder (Sch97, p.39)

```
1: G ← the graph of face-to-face connectivity of P
2: U ← breadth-first spanning tree of G.
3: return U
```

Algorithm A.7: The Depth-first Unfolder (Sch97, p.41)

```
1: G ← the graph of face-to-face connectivity of P
2: U ← depth-first spanning tree of G.
3: return U
```

Algorithm A.8: The Breadth/Depth Hybrid Unfolder

1: $G \leftarrow$ the graph of face-to-face connectivity of P
2: $U \leftarrow$ limited-depth-first spanning tree of G. New faces are added in groups by face neighborhood, groups are traversed breadth-first.
3: return U

The heuristic Unfolders - supporting subroutines

With the exception of the Star and Steepest Ascent unfolders, each of the progressive unfolders use heuristic tree search techniques. As has been discussed (Section 4.3,) heuristic tree search offers two primary ways to accelerate a search: *pruning* and *ordering*. These are implemented in two subroutines, each of which builds the net of the unfolding by gluing faces to available open edges. Both subroutines operate on a decision tree in which each node is a face to glue and each edge of the decision tree is a shared edge between two faces of P. In both routines, the pruning of the decision tree (be it immediate or deferred) provides implicit error correction and helps to avoid overlapped unfoldings.

`HeuristicMinimizer`(P, F_B, $Score()$) Maintains a stack of faces, the top of which is always the face to be expanded next; at each step of the unfolding, chooses the neighbor of the current face which has the lowest score. Prunes the decision tree by deferring nodes which would unfold to overlap, postponing them until they can be glued to another edge later in the unfolding process.

`SortingHeuristicMinimizer`(P, F_B, $Score()$) Maintains a list of available faces and a list of <face, edge> pairs which are known to unfold to overlap. These pairs represent edges of the decision tree which have already been pruned. At each step of the unfolding, selects the optimal (lowest-scored) available face and edge to glue next and expands the list of available faces with the neighbors of the newly-glued face.

As with the blind-search algorithms, these subroutines may leave some faces undeveloped if the faces persistently unfolded to overlap. A second pass is used to complete any (overlapped) unfolding.

Subroutine 3: `HeuristicMinimizer`(P, F_B, $Score()$)

1: $U \leftarrow \{F_B\}$ will be the unfolded net
2: $S \leftarrow \{F_B\}$ is the stack of developed faces to be expanded
3: **while** $S \neq \emptyset$ **do**
4: $f \leftarrow$ `Peek`(S) is the next face to expand
5: $L \leftarrow$ {the edges e_i of f where `GetNeighbor`$(f, e_i) \notin U$}
6: Sort L in descending order of `Score`(f_i, e_i)
7: $f' \leftarrow$ the first face in L such that gluing f' to f introduces no overlap
8: **if** $f' \neq \emptyset$ **then**
9: U+= $f \leftrightarrow f'$
10: `Push`(f', S)
11: **else**
12: `Pop`(S)
13: **end if**
14: **end while**
15: return U

Subroutine 4: `SortingHeuristicMinimizer`(P, F_B, $Score()$)

1: $U \leftarrow \{F_B\}$ U will be the unfolded net
2: $L \leftarrow$ {the neighbors of F_B} is the list of undeveloped faces to be expanded
3: $Blocked \leftarrow \emptyset$ is the list of <face, edge> pairs which are known to unfold to overlap
4: **while** $L \neq \emptyset$ **do**
5: $\{f, e\} \leftarrow$ the face f and its edge e which minimize `Score`(f, e), where $f \in L$, `GetNeighbor`$(f, e) \in U$, and $< F, e > \notin Blocked$
6: **if** $f = \emptyset$ **then**
7: break
8: **else if** gluing f at edge e introduces no overlap **then**
9: $L-= f$
10: U+= ($f \leftrightarrow$ `GetNeighbor`(`f`, `e`))
11: **else**
12: $Blocked$+=$< F, e >$
13: **if** all edges of f are in $Blocked$ **then**
14: $L-= f$
15: **end if**
16: **end if**
17: **end while**
18: return U

The heuristic Unfolders

Several of the heuristic Unfolders described here use the `HeuristicMinimizer` and `SortingHeuristicMinimizer` subroutines. To do so, each Unfolder redefines the function `Score`(f, e). `Score`(f, e) takes as parameters a face and an edge belonging to that face, and returns the 'score' of potentially gluing that face to the net. The heuristic subroutines then select faces based on lowest

score.

Algorithm A.9: The Least Height Unfolder

1: $F_B \leftarrow$ the lowest face in P
2: $Score(f, e) \leftarrow (\texttt{GetNeighbor}(f, e).center.Y + min(e_1.Y, e_2.Y))$
3: return $\texttt{SortingHeuristicMinimizer}(P, F_B, Score)$

Algorithm A.10: The Star Unfolder (Sch97, p.48)

1: $C \leftarrow$ an arbitrarily-chosen seed point on P
2: $U \leftarrow$ {the set of shortest edge-paths from C to each vertex v of P}
3: return U

Algorithm A.11: The Steepest Ascent Unfolder (Sch97, p.54)

1: $T \leftarrow \emptyset$ will be the output cut-graph
2: **for all** $v \in P$ **do**
3: $u \leftarrow$ the vertex in the 1-ring of v with the greatest Y co-ordinate
4: $T \mathrel{+}= (v \rightarrow u)$
5: **end for**
6: return U

Algorithm A.12: The Unravel Unfolder

1: $F_B \leftarrow$ the bottommost face of P
2: $Score(f, e) \leftarrow ($
3: $(\texttt{GetNeighbor}(f, e).center - f.center).normalized()$
4: dot-product with
5: $(((([0, 1, 0] \times f.normal) + [0, 5, 0]).normalized())$
6: $)$
7: return $\texttt{HeuristicMinimizer}(P, F_B, Score)$

The output-constrained heuristic Unfolders

All of the heuristic Unfolders described here re-define $\texttt{Score}(f, e)$ for use with `SortingHeuristicMinimizer`.

Algorithm A.13: The Convex Hull Unfolder

1: $F_B \leftarrow$ the lowest face in P
2: $Score(f, e) \leftarrow$ the area of the convex hull of the partial unfolding U if face f is glued at edge e
3: return $\texttt{SortingHeuristicMinimizer}(P, F_B, Score)$

Algorithm A.14: The Path Minimizer Unfolder

1: $F_B \leftarrow$ the lowest face in P
2: $Score(f, e) \leftarrow$ the number of edges crossed in the partial unfolding U between F_B and f
3: return $\texttt{SortingHeuristicMinimizer}(P, F_B, Score)$

Algorithm A.15: The Radius Minimizer Unfolder

1: $F_B \leftarrow$ the lowest face in P
2: $Score(f, e) \leftarrow$ the linear distance in the unfolded plane from the center of the development of F_B to the center of the development of f
3: return `SortingHeuristicMinimizer`$(P, F_B, Score)$

Unfolders for non-convex polyhedra

Any non-convex unfolding algorithm must be capable of returning failure (or an overlapped unfolding) because there are multiple known examples of nonconvex surfaces which cannot be unfolded. These algorithms must also be implemented more robustly than their convex-surface peers, as they may be asked to unfold surfaces with borders.

Each of the non-convex Unfolders described here assigns an average angle deficit to a face before unfolding. In the case of faces on a border of the polyhedron with vertices for which angle deficit cannot reasonably be calculated, the value -2π is used.

Algorithm A.16: The Curvature Ordering Unfolder

1: $F_B \leftarrow$ the face in P with highest average angle deficit
2: $Score(f, e) \leftarrow$ average angle deficit of the vertices of f
3: return `SortingHeuristicMinimizer`$(P, F_B, -Score)$ (Negated to minimize)

Algorithm A.17: The Tracer Unfolder

1: $F_B \leftarrow$ the face in P with highest average angle deficit
2: $Score(f, e) \leftarrow$ the absolute value of
3: $(f.center -$ `GetNeighbor`$(f, e)).normalized()$
4: dotted with
5: (the normalized gradient of the angle deficit at f)
6: return `HeuristicMinimizer`$(P, F_B, Score)$

Algorithm A.18: The Region Unfolder

1: The surface P is separated into distinct regions by positive (R_i^+) and negative (R_i^-) curvature.
2: The shortest paths ($S_{i\to j}$) on the surface between each positive region are recorded. Note that these paths must necessarily include faces of negative curvature.
3: Each positive region is independently unfolded to an isolated net, using a recursive call to another unfolding algorithm.
4: One positive region R_0^+ is arbitrarily designated as the 'root' of the unfolding.
5: The faces on the paths $S_{0\to i}$ from R_0^+ to each of the other positive regions are progressively glued to the unfolded net of R_0^+ until they reach R_i^+. The unfolded net of R_i^+ is then added to the unfolding, until all positive regions have been joined into a single unified unfolding U.
6: The border set of undeveloped faces around U then becomes the advancing wavefront of a second unfolding process, adding the negative regions onto the outer perimeter of the positive.

The Truncation Unfolder

The Truncation Unfolder is inspired by the leaf-truncation theorems of (BO07), but it carries the idea further by supporting truncation of surfaces beyond domes. As such the Truncation Unfolder as implemented here is not guaranteed to unfold a surface without overlap. For an example of why this is the case and further discussion of this algorithm, please see Section 2.4.

Algorithm A.19: The Truncation Unfolder

1: $A \leftarrow$ the highest vertex in P
2: $Q \leftarrow$ a pyramid with apex A, with one edge of the pyramid per edge incident to A on P
3: $T \leftarrow$ the non-base edges of Q will be the cut-graph
4: **while** $\|Q\| < \|P\|$ **do**
5: $\quad v \leftarrow$ the highest vertex on a face f in P which is not in Q
6: $\quad$ Truncate Q, taking its intersection with the infinite half-plane passing through f with normal opposite to the normal of f
7: $\quad$ Replace the edge in T which passed through v with an edge to v and two new edges from v to the base face along the two edges of f incident to v
8: **end while**

The Alpha-Beta Unfolder

For full discussion of the alpha-beta rules which define the behavior of the Alpha-Beta Unfolder, please see Section 3.3.

Algorithm A.20: The Alpha-Beta Rules

1: $C \leftarrow F_B.center$, the center of the development of root face F_B
2: $L \leftarrow \{$the vertices of $F_B\}$, a list of unexpanded vertices
3: $E \leftarrow$ a list of expanded vertices, initially empty
4: **while** L is not empty **do**
5: $v \leftarrow$ the vertex in L whose furthest development is closest to C
6: **if** v is the tip of a cut-path Φ **then**
7: $w \leftarrow$ the vertex in the 1-ring of v such that $\beta_w > (\pi - \alpha_v)/2$ and which maximizes $(v^I - C \cdot w^I - v^I)$
8: **else**
9: $w \leftarrow$ the vertex in the 1-ring of v which maximizes $(v - C \cdot w - v)$
10: **end if**
11: Cut the edge $v \rightarrow w$
12: Develop all faces around v and append to L all vertices on the new perimeter which are not already in L or E
13: Remove v from L, add v to E
14: **if** $w \in E$ **then**
15: $w' \leftarrow w$
16: **while** $w' \in E$ **do**
17: $temp \leftarrow$ the vertex to which w was cut
18: Remove w from E, add w to L
19: $w' = temp$
20: **end while**
21: **end if**
22: **end while**

The Collision Repair Unfolder

The Collision Repair Unfolder is described in detail in Section 3.4. It relies on subroutine 5, `Repair`(C), to repair each individual collision. Subroutine 5 is reprinted from Algorithm 3, Section 3.4.

Subroutine 5: `Repair`(C)

1: $X \leftarrow$ the first crossed edge in the collision C
2: $Y \leftarrow$ the edge furthest upstream which crosses X
3: $P_X \leftarrow$ the polygon in the source polyhedron whose development contains the edge X
4: $P_Y \leftarrow$ the polygon in the source polyhedron whose development contains the edge Y
5: $L \leftarrow \{p_0 = P_X, p_1, \ldots, p_{n-1} = P_Y\}$ is the shortest path on the polyhedron from P_X to P_Y
6: **for** each $p_i \in L$, $i > 0$ **do**
7: $e \leftarrow$ the edge shared between p_{i-1} and p_i
8: $g \leftarrow$ the edge shared between p_i and the parent of p_i
9: **if** $e \in T$ **then**
10: $G-= e$
11: $G+= g$
12: **end if**
13: **end for**

Algorithm A.21: The Collision Repair Unfolder

1: $U \leftarrow$ a Steepest Edge unfolding of P
2: **while** U has conflicts **do**
3: $C \leftarrow$ the conflict which lies closest to the center of the unfolding
4: `Repair`(C)
5: **end while**
6: return U

Appendix B

YAMM

The YAMM[1] software package provides a forum for experimentation with unfolding. It allows the user to view three-dimensional polyhedral models, cut their edges by various unfolding algorithms, and animate the results. YAMM has been developed in C++ using Microsoft Visual Studio 6.0 and 8.0 under Windows XP and Vista. A compiled executable may be downloaded from

`http://bentonian.com/Papers/Dissertation/`

and the source code is available for academic use upon request.

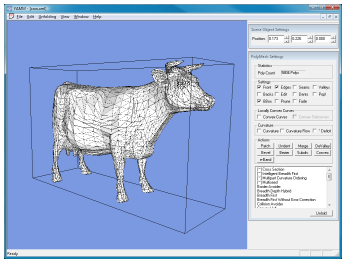
This Appendix is provided for the reader as an introduction to the features of YAMM. It is not intended as a complete survey of the software, nor as a user's manual. For further details on YAMM, please consult the author directly.

B.1 Principal Features

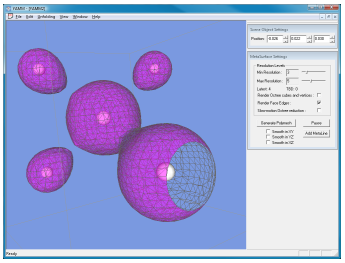
YAMM offers the following principal features, among others; a few screen-shots are presented in figure B.1:

- Polyhedral mesh modeling
  - Implicit surface modeling with metaballs and metalines
  - Bézier patch modeling (in a limited fashion)
  - Subdivision (in a limited fashion)
  - Precomputed mathematical models chosen for interest, including two different ways to generate random convex polyhedra
  - Manual vertex and face editing
- Unfolding of polyhedral surfaces through a large suite of unfolding algorithms, which can be extended through the addition of further modules written in C++
- Animation of unfoldings
- Curvature and convex hull analysis

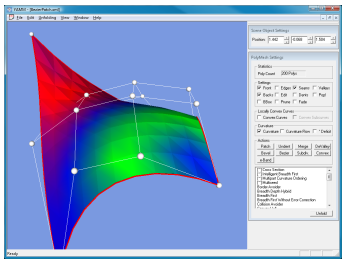
[1] 'YAMM' stands for 'Yet Another Metaball Modeler'; the software was originally designed as an implicit surface modeling tool. It has outgrown its roots.



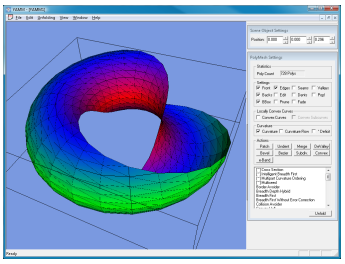
(a) The cow PolyMesh



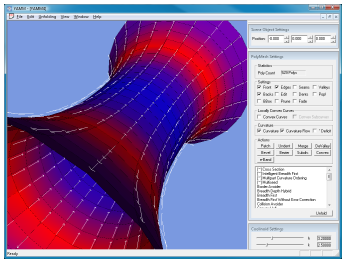
(b) Implicit surface modeling



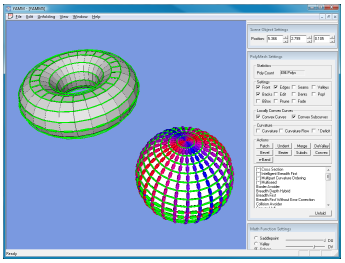
(c) A Bézier patch, with curvature



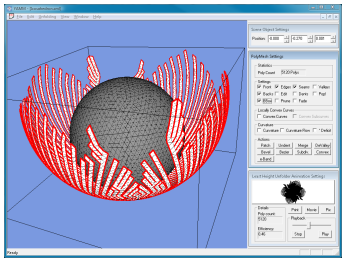
(d) A twisted half-torus



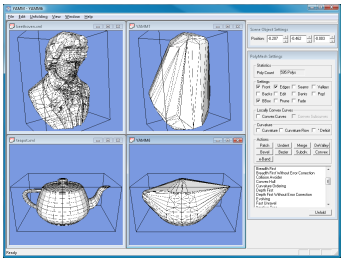
(e) Curvature flow on a cooli-noid



(f) Convex curves



(g) Unfolding a geodesic sphere



(h) Finding convex hulls

Figure B.1: YAMM in action

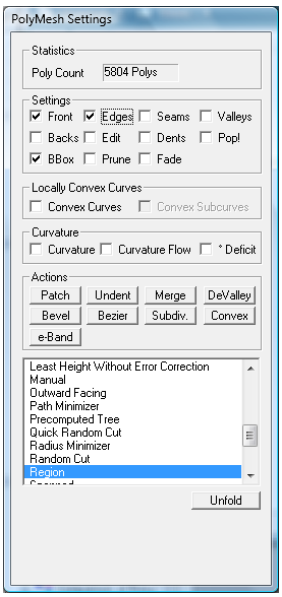


Figure B.2: The PolyMesh settings panel provides a host of rendering, editing and unfolding options for the PolyMesh. Options include:
◦ Back- or front-face culling; edge highlighting; alpha-blending
◦ Show or hide bounding box, seams, valleys and dented faces
◦ Edit vertices
◦ 'Pop' (extrude) faces
◦ Highlight convex curves
◦ Patch seams, remove dents, merge coplanar convex faces and flip valley edges
◦ Bevel corners, replace faces with Bézier patches, and subdivide faces
◦ Find the convex hull
◦ Replace every triangle with a polyhedral banding hexagon
◦ A selection of Unfolders

- Recording to still image and video
- ...and many more!

B.2 Modeling Polyhedra

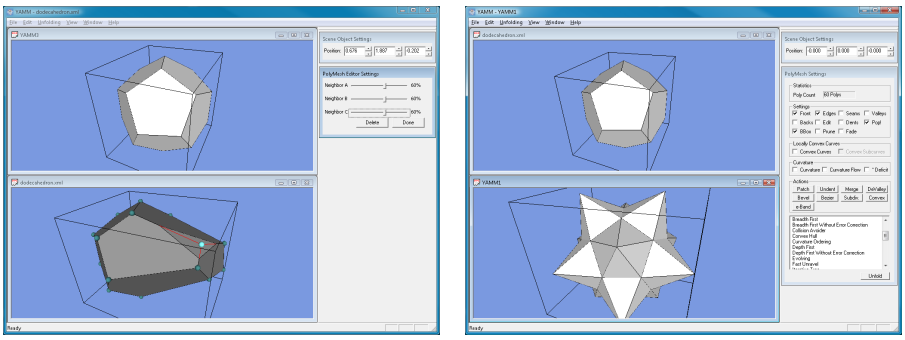
YAMM's internal representation of a polyhedron is called a *PolyMesh*. Many of the features of YAMM are accessed through manipulation of a PolyMesh; Figure B.2 shows the PolyMesh settings panel and gives a brief overview of the features available.

A PolyMesh stores the lion's share of its data in a linked list of vertices and a linked list of faces; faces store counter-clockwise-ordered lists of pointers to vertices and vertices stores unordered lists of faces.

B.2.1 Vertex editing and mesh repair

While YAMM makes no pretense at being a 3D editing platform, it does offer a few tools which can be useful in constructing test cases and modifying small models. Users can edit individual vertices, dragging them in 3D or truncating them by using sliders to choose a cutoff plane (Figure B.3(a).) Users can also 'pop' faces, replacing a face with a pyramid whose height is a function of the size of the face (Figure B.3(b).) This allows a user to, for example, quickly stellate a polyhedron.

YAMM also has several features which were built to support ill-behaved geometry, a common issue when models are downloaded from online sources. Seams (edges which are reported as border edges, having only one attached face, which actually overlap another edge in the same position believed to be a border from the other side) can be 'patched' and spliced together at the click of a button. Coplanar faces which together form a convex boundary can be merged



(a) Editing vertices manually, truncating one vertex

(b) 'Popping' the faces of a dodecahedron

Figure B.3: Editing PolyMeshes

into a single convex face, a useful feature when downloaded data is undesirably simplicial. It is also not uncommon for surfaces which are generally convex to be locally concave because a square has been split into two triangles with the diagonal edge creating a valley instead of a ridge; this can be detected by comparing normals and, in many cases, the square re-split with the other diagonal following a ridge line. These automated repair features can be very useful when building a test-case in, for example, YAMM's implicit-surface modeling tool.

B.2.2 Beveling, banding, and 'bezification'

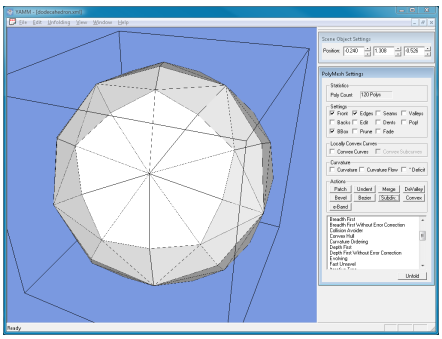
YAMM implements several automated tools to convert a PolyMesh into a more complex (and hopefully more interesting) PolyMesh. Chief among these are:

Subdivision Every non-simplicial face on the surface is triangulated. Then every triangle is replaced with four triangles, joining the midpoints of the edges of each source face; midpoints are shifted away from the center of the surface to have radius equal to the average of the radii of the endpoints of the edge. This is an excellent way to convert, for example, an icosahedron into a geodesic sphere (Figure B.4(a).)

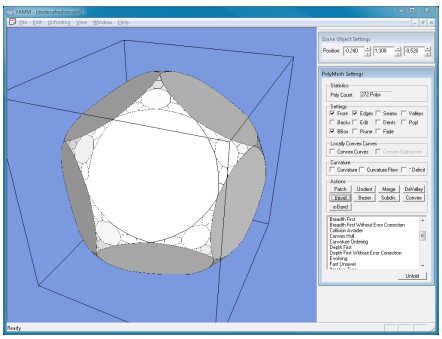
Beveling Every vertex on the convex hull of the surface is truncated, by a plane which passes through the edges incident to the vertex at 1/3 their length (or the closest best-fit match) (Figure B.4(b).)

Banding Each face of the polyhedron is 'banded' by the polyhedral banding algorithm (Algorithm 2), replacing each k-sided face with $1+2k$ new faces (Figure B.4(c).)

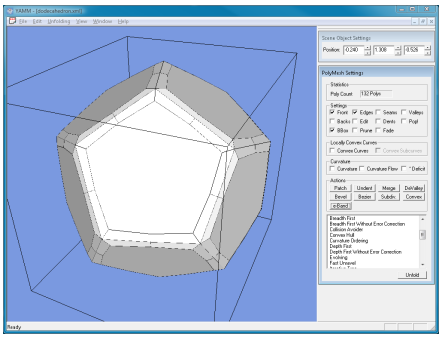
Béziers Each face on the surface is replaced by an appropriate combination of triangular and quadrilateral Bézier patches. (Figure B.4(d).)



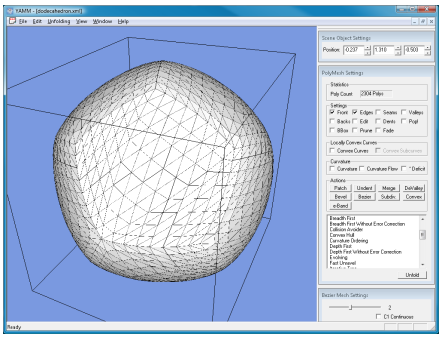
(a) Triangle subdivision



(b) Beveled vertices (beveled three times)



(c) Polyhedral banding



(d) Bézier patches

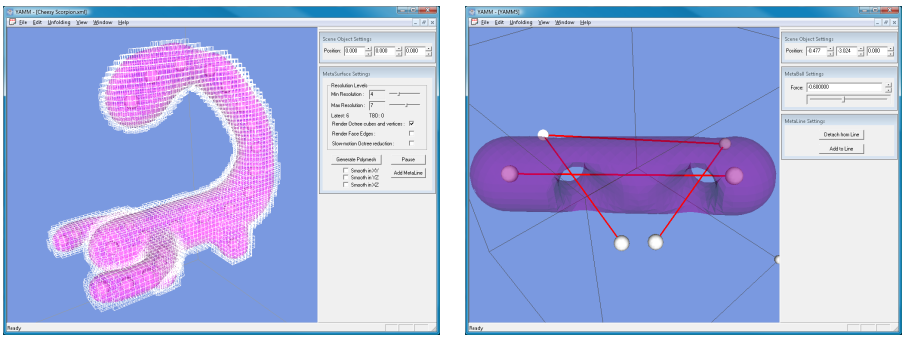
Figure B.4: Advanced operations on a dodecahedron

B.3 Implicit Surface Modeling

The YAMM engine was first developed as an implicit surface modeler and it supports those early features to this day. To discuss implicit surface modeling, or 'metaball' modeling as Jim Blinn called it, is well outside the scope of this document; the reader is referred to (Bli82), which is more or less the seminal work in the field, and to Paul Bourke and Marco Pugliese's very readable online overview and history of the field (Bou97).

YAMM supports *metaballs* and *metalines*. The user controls the scalar force on the meta-primitive, including negative values (Figure B.5(a)) and, with patience, can model simple shapes (Figure B.5(b).) Implicit surfaces are found with recursive octree refinement (shown as cubes in Figure B.5(b)) which allows complete control over the smoothness of the mesh of faces. The implicit surfaces generated can then be converted into PolyMeshes for editing and unfolding.

YAMM also offers software support for *metasurfaces*: classes derived from the metaball class which generate their force functions from other procedural data. This is how the implicit surfaces in Chapter 5 are created: the unfoldability of a coolinoid, for a given h, k and dim, is sampled as the force function of an implicit surface being evaluated over a matching x,y,z domain. An overlap-free



(a) Simple blobby modeling (b) Negative metalines

Figure B.5: Advanced operations on a dodecahedron

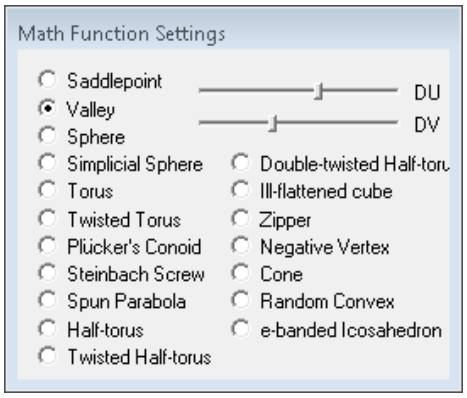


Figure B.6: MathFn settings panel

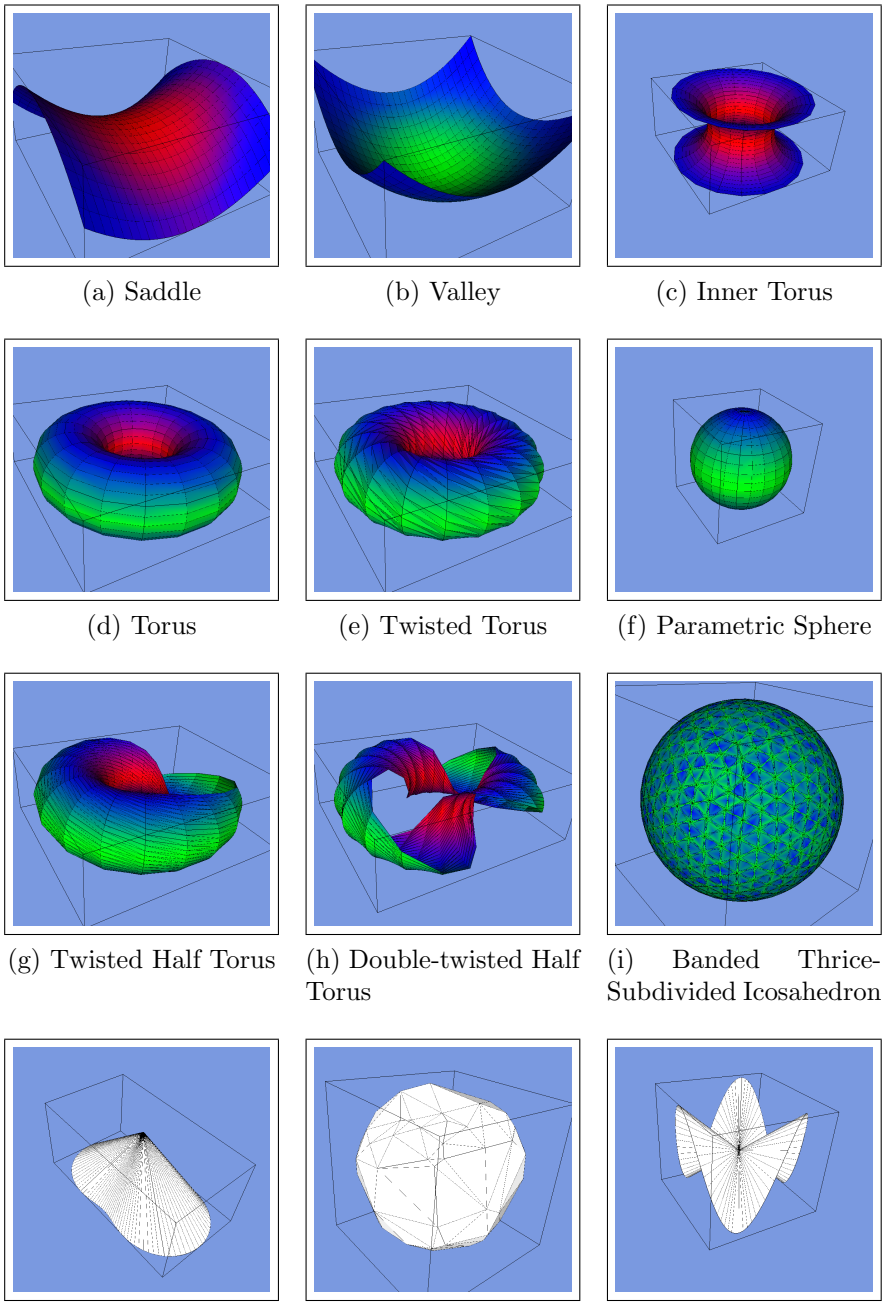
unfolding maps to 1, self-intersection to 0, and the resulting surface interpolates the boundary of 0.5.

In YAMM, an implicit surface can be added from the Edit menu or by pressing the F2 key.

B.4 MathFn

One particularly useful class derived from PolyMesh is the MathFn object, which hardcodes a set of frequently-used primitives for easy access. The MathFn settings panel (Figure B.6) lets the user choose from 21 preset test models, tuned by two scalar slider controls. A sampler of the presets are presented in Figure B.7.

The *du* and *dv* sliders usually set the resolution of the surface; for example, in the case of the parametric sphere, *du* determines the number of quads in one horizontal ring on the surface and *dv* determines the number of rows of quads. In the case of the random convex polyhedron, *dv* is unused and *du* sets an approximate vertex count.



(a) Saddle (b) Valley (c) Inner Torus

(d) Torus (e) Twisted Torus (f) Parametric Sphere

(g) Twisted Half Torus (h) Double-twisted Half Torus (i) Banded Thrice-Subdivided Icosahedron

(j) Spherical Vertex (k) Random Convex Polyhedron (l) Monkey Saddle

Figure B.7: A few of the preset models built into the MathFn object

In YAMM, a MathFn object can be added from the Edit menu or by pressing the F1 key.

B.5 Unfolding

When a PolyMesh object is selected in YAMM, the properties panel includes a list of available Unfolders. Double-clicking on any Unfolder will run that algorithm on the chosen mesh. Most unfolders work progressively and their results can be monitored in the 3D view or observed in 2D projection in the properties panel.

The Unfolders module of YAMM is designed to be an extensible library of experimental unfolding algorithms. All Unfolders derive from a base class, which runs the core unfolding loop; new unfolders are added through a typical Factory design pattern. Every Unfolder must be able to answer a few basic queries:

- Is it done yet?
- When done, what is the ratio of the number of non-overlapped faces to the total number of faces?
- What is the current net? Current cut-graph?
- Is it generating more than one unfolding, and if so,
  - How many does it expect to generate?
  - How many so far?
  - How many have been successful (i.e., free of overlap?)

When an unfolded net is assembled, it is usually placed 10% of the source model's height below the lowest face on the model and rotated so that the lowest faces roughly match in position and orientation. This lends an intuitive link between source and net. The unfolding is actually a PolyMesh in its own right; converted to an animation, it can be manipulated with all the tools that control and edit the PolyMesh.

Internally an unfolding stores its connectivity in the data structures of PolyMesh, augmented by a tree of matrix transforms relative to the 'root face' (usually the lowest face in the source polyhedron.) In an earlier design every face stored its vertex data separately and computed its final position in the plane by calculating its rotation around the anchoring edge that connected it to the root, but this was prone to floating-point drift; calculating a fresh projection every time accumulated imprecision and sometimes produced nonplanar unfoldings. The current design stores a tree of matrices which matches the unfolded net of the PolyMesh; each node of the transform tree carries the matrix transform necessary to rotate the corresponding face into the plane of its parent in the unfolded net. This tree of matrices is much more accurate and less vulnerable to floating-point error.

Visually, the unfolded net can be used to display meta-data about the unfolding as well. Collisions are highlighted by the familiar red-shaded polygons. The BSP (Binary Space Partitioning tree, discussed below) shows the collision-detection data structures which support the unfolding. The user can request

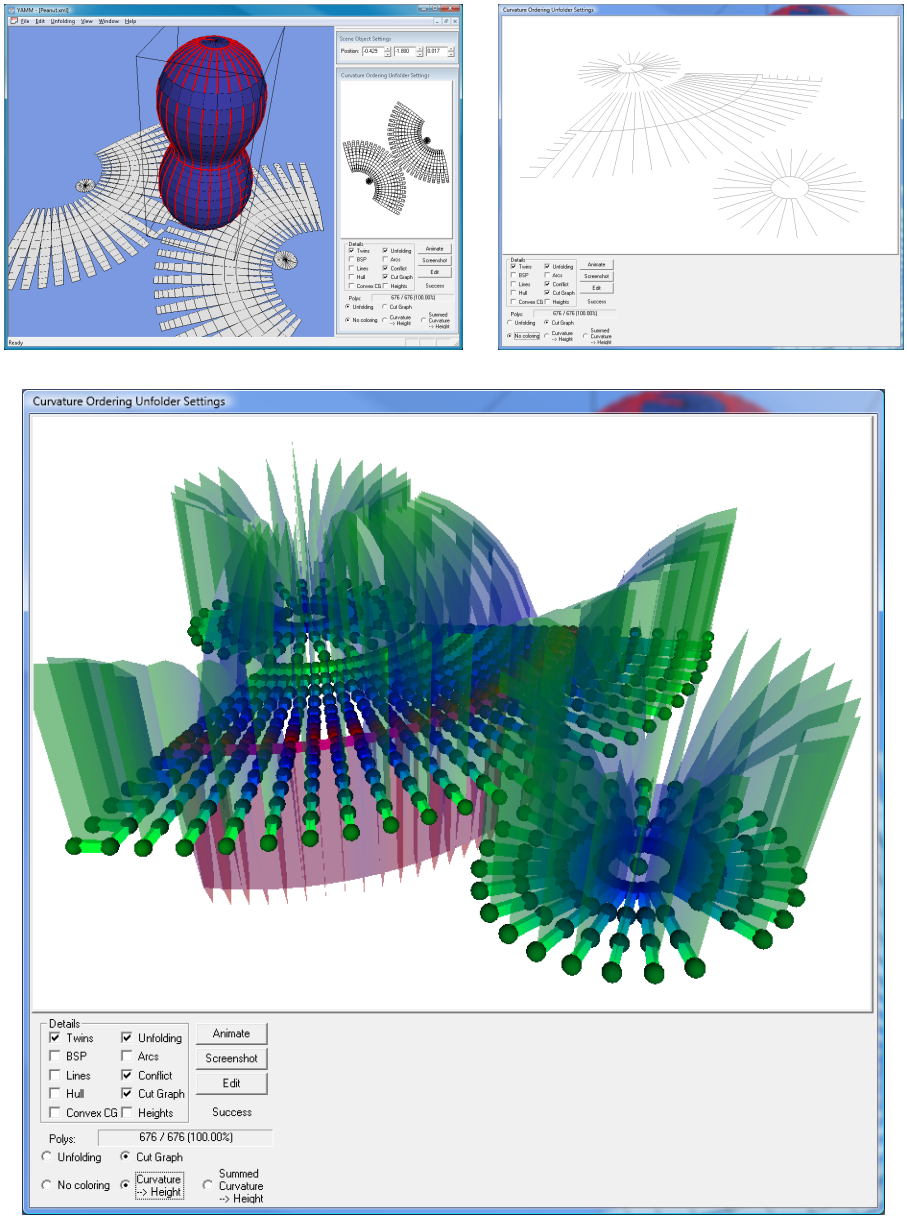


Figure B.8: (a) Unfolded net for the peanut model (b) Cut-graph for the peanut (c) The peanut cut-graph shaded and height-mapped by curvature. Notice how the open spaces between regions of the cut-graph are bordered by the vertices of greatest curvature: where there is a gap in the cut-graph, there are no cuts in the net. The Curvature Ordering Unfolder unfolds outwards from the faces with highest average curvature, and so they are surrounded by un-cut edges.

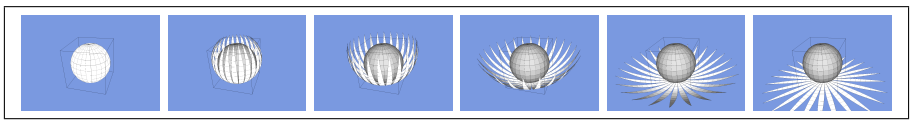


Figure B.9: The animated unfolding a sphere

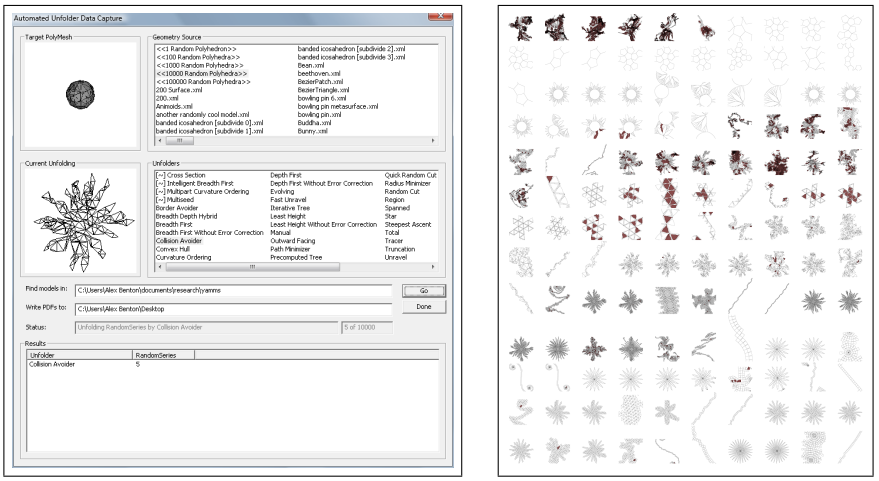


Figure B.10: (a) The Unfold-A-Matic dialog (b) Auto-generated unfoldings

that the arc of each leaf-node of the cut-graph be rendered; likewise the convex hull of the unfolded net; and the entire unfolding can even be shaded by the height of each vertex (a useful visualization for Unfolders like Least Height and Collision Repair.) Another very useful feature is that any unfolding can be manually edited, which is very useful for exploring the effects of different cut-graphs on the final net.

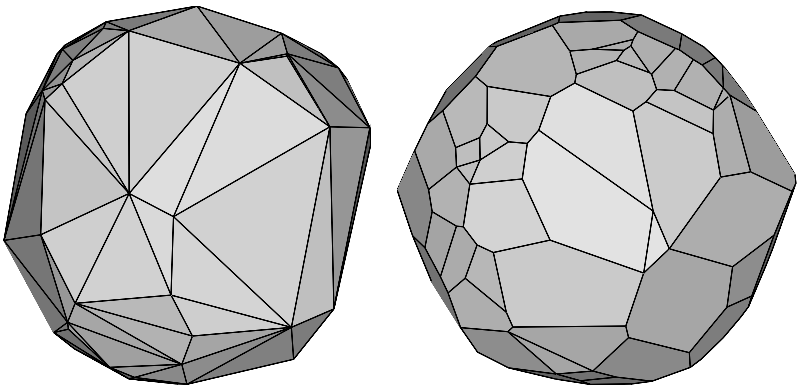
The property panel for an unfolding can offer a wealth of information. It can display both the unfolded net (Figure B.8(a)) and the cut-graph (Figure B.8(b)). A secondary mode when viewing the cut-graph will project virtual space above the cut-graph to show the angle deficit at each vertex in the graph, shaded by magnitude and direction of curvature (Figure B.8(c).)

Any Unfolder can also be converted into an animation, for slideshow or movie output (Figure B.9.)

B.6 Automated data collection

The Unfold-A-Matic dialog (Figure B.10) automates data collection by loading large numbers of models for unfolding by different algorithms and recording the data to a spreadsheet.

These data sets can be huge, stretching to up to 100,000 randomly-generated convex polyhedra at a time. With early Unfolders this was never necessary but with more recent techniques–the Least Height Unfolder and Collision Repair–sometimes 100,000 polyhedra of approximately 300 vertices apiece, or more, are



(a) (b)

Figure B.11: Generating random convex polyhedra: (a) Method one (128 faces) (b) Method two (144 faces)

necessary to construct a statistically significant sample.

To support a Monte-Carlo method of finding the average unfoldability of a given surface the Unfold-A-Matic offers a brute-force mode, in which random unfoldings are generated continuously. It should be noted, however, that this is not an optimal form of randomized simulation: the cut-graphs generated are generated separately each time and do not avoid duplicates, which over time can lead to redundancy and a diminishing of the rate at which new unfoldings are found. This does not impact the probability distribution of the results discovered but it does reduce–sometimes significantly–the speed with which they are determined.

B.7 Interesting Internal Algorithms

B.7.1 Generating random polyhedra

Generating a random convex polyhedron is not as easy as one might think. Two methods are commonly used for generating a random closed convex surface of roughly unit radius:

1. Take the convex hull of a set of randomly-generated set of points at unit distance from the origin (Figure B.11(a).)

2. Take the intersection of a set of randomly-generated infinite half-planes each of which passes through a randomly-chosen point at unit distance from the origin with normal equal to the negation of the point (Figure B.11(b).)

These two methods produce notably dissimilar outputs. In method one, the odds of more than three vertices happening to be exactly coplanar are extraordinarily low; therefore the polyhedra produced are almost entirely simplicial. In contrast, in method two it is very uncommon that more than three planes intersect at

Algorithm 6: Enumerating spanning trees

```
1: L ← the list of edges of G, ordered such that each edge other than the
   first edge in the list shares an endpoint with an edge before it in the list;
   L is the list of edges to be added to the evolving list of spanning trees
2: T ← a list of one element, which is a tree of one element, which is the
   first edge popped off of L; T will store the evolving list of spanning trees
3: for all e ∈ L do
4:   T′ ← ∅
5:   for all t ∈ T do
6:     if Only one endpoint of e is in t then
7:       T′ += (t + e)
8:     else
9:       C ← the loop of edges in the tree t from one endpoint of e to the
         other
10:      for all e_C ∈ C do
11:        T′ += (t + e − e_C)
12:      end for
13:    end if
14:  end for
15:  T ← T′
16: end for
```

a single point, and so trivalent vertices predominate. The two methods are compared side-by-side in Figure B.11.

YAMM uses method one for most Monte-Carlo simulations, primarily because method one is significantly faster to compute with the software available.

B.7.2 Enumerating spanning trees

A number of algorithms have been proposed for enumerating the spanning trees of a graph; Malcolm Smith gives a well-documented survey of the field in his 1997 MSc thesis (Smi97). It appears that the best method currently known is that of Kapoor, Kumar and Ramesh (KR00), but YAMM does not use this algorithm. To support the Family Tree Unfolder, the model used by Kapoor et al. was insufficient: the author needed a more iterative algorithm to list spanning trees, which would allow snapshots of state to be taken on an edge-by-edge basis, and so a much slower (but much simpler, and very easy to code) method was adopted. This lack of optimal implementation should not be seen as a failure on YAMM's part; the enumeration of spanning trees is a 'black box' problem, independent of unfolding and well outside the scope of YAMM and this dissertation.

The method used in YAMM to enumerate the spanning trees of a graph G is given in Algorithm 6.

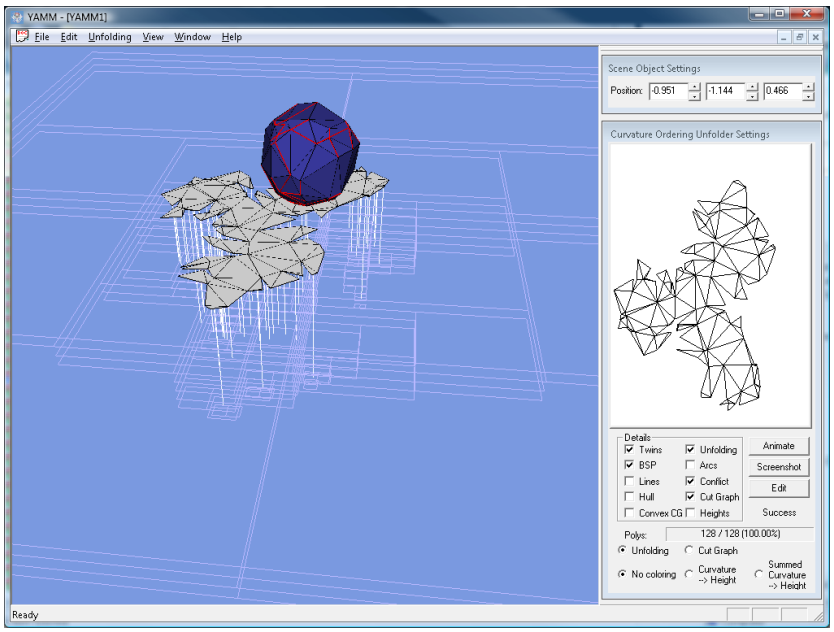


Figure B.12: The BSP of an unfolding is shown by a series of nested rectangles subdividing the horizontal plane. Vertical lines connect each unfolded face to the node in the tree which contains it.

B.7.3 Binary Space Partitioning trees

A *binary space partitioning tree* ('BSP') is a data structure often used in computer graphics to accelerate spatial lookup. A BSP tree is a subdivisions of space; each non-leaf node stores a plane and possibly a few points, each leaf node stores a list of points. Each non-leaf node has two children, named 'left' and 'right'. Any point stored in the 'left' child will always fall on one side of the plane; points stored under the 'right' child will fall on the other side; and points which are exactly in the plane of the node are listed in the node itself.

The BSP accelerates spatial lookup for operations such as finding all points in a set within a certain radius. This is an essential optimization for detecting collisions between faces in an unfolding. Consider this: unaccelerated, to test for a collision between some new triangle f and a set F of n triangles which have already been unfolded could require as many as $3 \times 3 \times n$ edge-intersection tests. Using a BSP, an $O(\log n)$ search will return the very short list of faces (usually just three or four) in F which lie close enough to f to possibly intersect; to test that short list is a very significant improvement in time.

B.7.4 Dijkstra's Algorithm and shortest paths

YAMM implements Dijkstra's Algorithm (Dij59) for finding the shortest paths on a graph. To find shortest paths across faces on a polyhedron, Steiner point interpolation is used, inspired by (ALMS98). As Aleksandrov et al. demonstrate, the placement and distribution of Steiner points can be heavily optimized, but

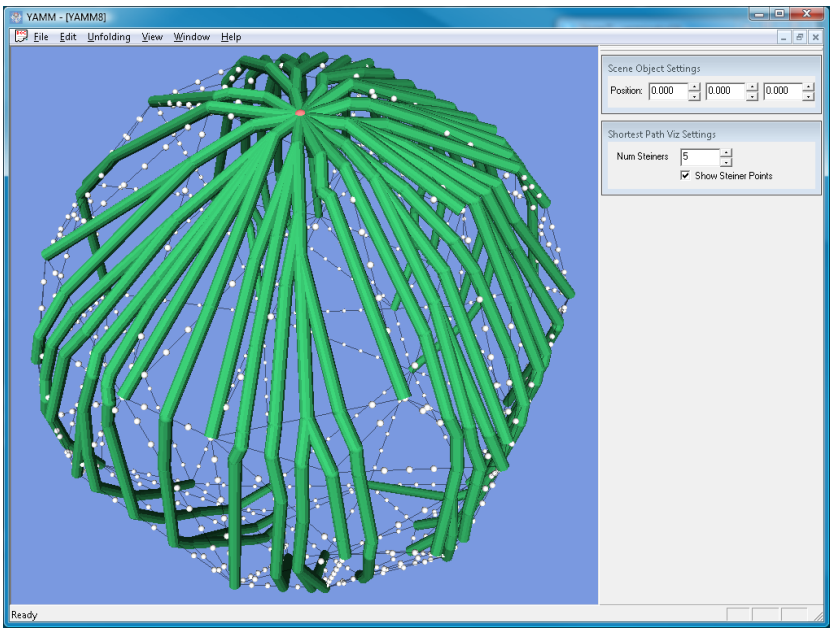


Figure B.13: Shortest paths approximated for a minimum of five evenly-spaced Steiner points per edge on a randomly-generated convex polyhedron

the implementation in YAMM does not leverage many of their techniques. A sample of shortest paths from a point on a random convex polyhedron to every vertex is shown in Figure B.13.

B.8 Data Formats

YAMM accepts the following file types as input, with varying degrees of feature support:

- XML file format (.xml)
- 3D Studio Max models (.3ds and .asc)[2]
- OFF files (.off)[3]
- VRML 1.0 and 97 models (.wrl)[4]

YAMM supports the following forms of data export:

- XML file format (.xml); includes support for color, texture, and camera settings
- OFF files (.off)

[2] `http://www.fileformat.info/format/3ds/`
[3] `http://people.scs.fsu.edu/~burkardt/data/off/off.html`
[4] `http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/`

YAMM offers the following forms of data capture:

- Capture images to .png with alpha-blended transparency
- Capture images to .pdf with vector-rendered PostScript annotations
- Capture animations to .png slideshow or .avi movie
- Automated unfolding of large sets of models by large sets of Unfolders, with spreadsheet recording for the results
- Automated coolinoid unfoldability discovery through octree reduction

Most data files are stored in YAMM's internal XML structure, an extensive XML hierarchy which will not be detailed here. The use of XML lends flexibility, legibility and easy manual adjustment of data co-ordinates. It also means that fields can be added or removed from the file without loss of use, and that data copied to the clipboard can be read in any text editor.

PolyMesh data is stored in the traditional "list of vertices, list of faces as indices" model. Vertices are sometimes tagged with an additional string, '[bin]'. This optional string is a hex transcription of the binary representation of the three double-precision floating-point values which store the vertex co-ordinates. For high-resolution models, the loss of precision from truncating floats to four significant digits had proved unacceptable.

A typical XML data file for YAMM might look like this:

```
<?xml version="1.0" ?>

<SceneRoot>
  <Children>
    <PolyMesh>
      <Vertices>
        <Vertex> 1.0000  1.0000  1.0000</Vertex>
        <Vertex>-1.0000 -1.0000  1.0000</Vertex>
        <Vertex>-1.0000  1.0000 -1.0000</Vertex>
        <Vertex> 1.0000 -1.0000 -1.0000</Vertex>
      </Vertices>
      <Faces>
        <Face>  0,  2,  1</Face>
        <Face>  0,  1,  3</Face>
        <Face>  0,  3,  2</Face>
        <Face>  1,  2,  3</Face>
      </Faces>
      <DrawEdges>true</DrawEdges>
      <Color>0.7500  0.7500  0.7500</Color>
    </PolyMesh>
  </Children>
  <Viewpoint>
    <At>   0.0000  0.0000  0.0000</At>
    <From> 5.4875  6.6461  5.0711</From>
    <Up>  -0.7816  0.6232  0.0290</Up>
  </Viewpoint>
</SceneRoot>
```

Appendix C

Models

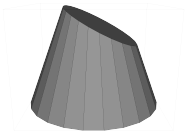
Many of the models used in this dissertation have been made available in an online repository. Many can be downloaded in .OFF format for use in most popular 3D software, and all can be downloaded in .XML format for use in YAMM. The models are available at

`http://bentonian.com/Papers/Dissertation/`

The files *Bunny*, *Cow*, *Duck*, and *Teapot* are not original creations. Each of these models is publicly available on the internet. The *Fukuda's Tetrahedron* model is this author's implementation of Fukuda's original surface.

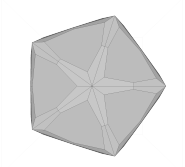
Convex surfaces

Dome

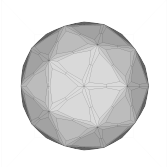


Mesa

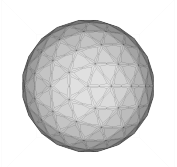
Fukuda's Tetrahedron



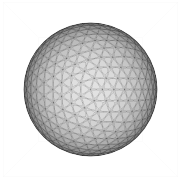
Banded Icosahedron [Level 0]



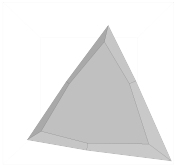
Banded Icosahedron [Level 1]



Banded Icosahedron [Level 2]

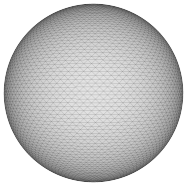


Banded Icosahedron [Level 3]

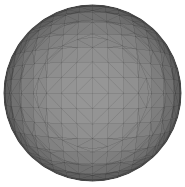


Banded Tetrahedron

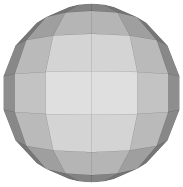
C.0.1 Spheres



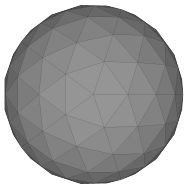
Hi-resolution Simplicial



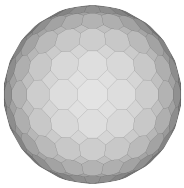
Octrees



Parametric



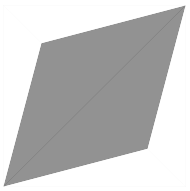
Simplicial



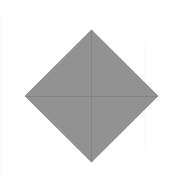
Trivalent

C.0.2 Platonic solids

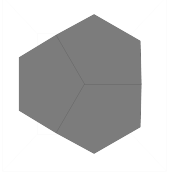
The Platonic Solids



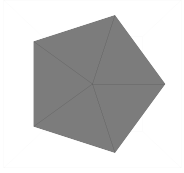
Tetrahedron



Octahedron

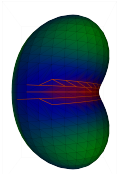


Dodecahedron

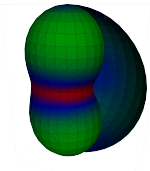


Icosahedron

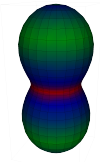
C.0.3 Non-convex surfaces



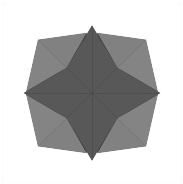
Bean



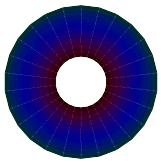
Curl



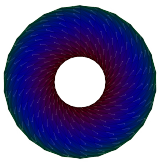
Peanut



Star



Torus

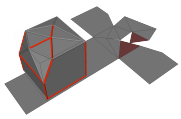


Twisted Torus

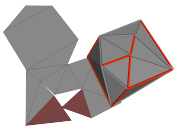
C.0.4 Examples and counterexamples

Empty Sector

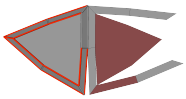
Empty Sector (trimmed)



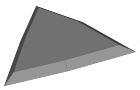
Local Convexity



Local Convexity (minimal)



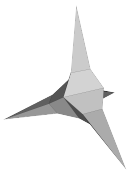
Prismatoid banding



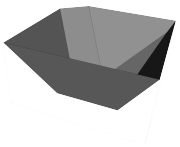
Prismatoid banding (doubled)

Wide Pyramid

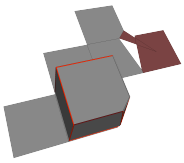
Wide Pyramid (truncated)



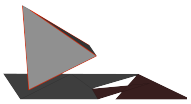
Witch's Hat Assembly



Convex Cap

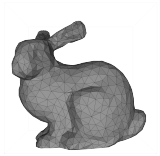


Truncated Cube (with conflict)



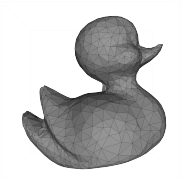
Truncated Tetrahedron (with conflict)

C.0.5 Scanned models

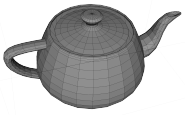


Bunny

Cow

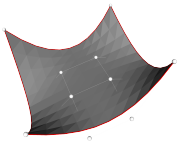


Duck

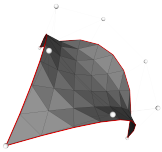


Teapot

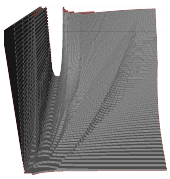
C.0.6 YAMM models and implicit surfaces



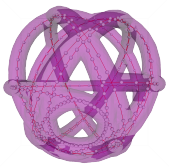
Bézier Patch



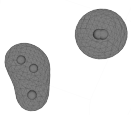
Bézier Triangle



Coolinoid Developability



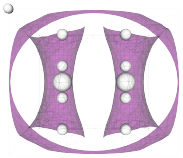
"200"



"Animoids"

"Bowling Pin"

“Egg”



“Randomly Cool Model”

“Scorpion”

Citations

[AAOS90] Pankaj K. Agarwal, Boris Aronov, Joseph O'Rourke, and C. Schevon. Star unfolding of a polytope with applications. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes Comput. Sci.*, pages 251–263. Springer-Verlag, 1990.

[ADL+04] Greg Aloupis, Erik D. Demaine, Stefan Langermann, Pat Morin, Joseph O'Rourke, Ileana Streinu, and Godfried Toussaint. Unfolding polyhedral bands. In *Proc. 16th Canad. Conf. Comp. Geom.*, pages 60–63, 2004.

[Ale61] Paul Alexandroff. *Elementary Concepts of Topology.* Dover Publications Inc, New York, 1961. ISBN 0-486-60747-X.

[ALMS98] Lyudmil Aleksandrov, Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. An epsilon-approximation for weighted shortest paths on polyhedral surfaces. In *Scandinavian Workshop on Algorithm Theory*, pages 11–22, 1998.

[AO91] Boris Aronov and Joseph O'Rourke. Nonoverlap of the star unfolding. Technical Report 002, Smith College, Northampton, MA, Mar 1991.

[BCO04] Nadia Benbernou, Patricia Cahn, and Joseph O'Rourke. Unfolding smooth prismatoids. In *Proc. 14th Annu. Fall Workshop Comput. Geom.*, pages 12–13, November 2004.

[BDE+03] Marshall Wayne Bern, Erik D. Demaine, David Eppstein, Eric Heng-Shiang Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry Theory & Applications*, 24(2):51–62, February 2003.

[Ben06] Alex Benton. Angular criteria for unfoldable convex polyhedra. In Patrick Chenin, Tom Lyche, and Larry L. Schumaker, editors, *Curve and Surface Design: Avignon 2006*, pages 41–47, 2006.

[Ben07] Alex Benton. A developable surface of uniformly negative internal angle deficit. In Ralph R. Martin, Malcolm A. Sabin, and Joab R. Winkler, editors, *IMA Conference on the Mathematics of Surfaces*, volume 4647 of *Lecture Notes in Computer Science*, pages 64–77. Springer, 2007. ISBN 978-3-540-73842-8.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[BO07] Alex Benton and Joseph O'Rourke. Unfolding polyhedra via cut-tree truncation. In *Proceedings of the 19th Annual Canadian Conference on Computational Geometry (CCCG 2007)*, pages 77–80. Carleton University, Ottawa, Canada, August 2007.

[BO08] Alex Benton and Joseph O'Rourke. A class of convex polyhedra with few edge unfoldings. Technical Report 089, Smith College, Jan 2008. http://arxiv.org/abs/0801.4019.

[Bou97] Paul Bourke. Implicit surfaces, 1997. `http://ozviz.wasp.uwa.edu.au/~pbourke/modelling\_rendering/implicitsurf/`.

[DEE+02] Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. Vertex-unfoldings of simplicial manifolds. In *18th Annu. ACM Sympos. Comput. Geom.*, pages 237–243, 2002.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DO05] Erik D. Demaine and Joseph O'Rourke. A survey of folding and unfolding in computational geometry. In Jacob E. Goodman, János Pach, and Emo Welzl, editors, *Combinatorial and Computational Geometry*, volume 52 of *Mathematics Sciences Research Institute Publications*, pages 167–211. Cambridge University Press, 2005.

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521857570.

[Dür25] Albrecht Dürer. *The Painter's Manual, translated by W.L. Strauss (1977)*. Abaris Books, 1525. ISBN 0913870528.

[Eps05] David Epstein. The geometry junkyard - nineteen proofs of Euler's formula, 2005. `http://www.ics.uci.edu/~eppstein/junkyard/euler/`.

[Fuk97] Komei Fukuda. Strange unfoldings of convex polytopes, 1997. `http://www.ifor.math.ethz.ch/~fukuda/unfold_home/unfold_open.html`.

[GBKK98] S.K. Gupta, D.A. Bourne, K. Kim, and S.S. Krishnan. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[Gri76] H.B. Griffiths. *Surfaces*. Cambridge University Press, 1976. ISBN 0521206960.

[HP04] Michael Hofer and Helmut Pottmann. Energy-minimizing splines in manifolds. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 284–293, New York, NY, USA, 2004. ACM.

[KR00] Sanjiv Kapoor and H. Ramesh. An algorithm for enumerating all spanning trees of a directed graph. *Algorithmica*, 27(2):120–130, 2000.

[Lak76] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976. ISBN 0-521-29038-4.

[LMS97] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating weighted shortest paths on polyhedral surfaces. In *6th Annual Video Review of Computational Geometry, Proc. 13th ACM Symp. Computational Geometry*, pages 485–486. ACM Press, 4–6 1997.

[Loo94] B. Van Loon. *Geodesic Domes*. Tarquin Press, Feb 1994. ISBN 0906212928.

[Luc06] Brendan Lucier. Unfolding and reconstructing polyhedra. Master's thesis, University of Waterloo, 2006.

[MDSB02] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. Discrete differential geometry operators for triangulated 2-manifolds, 2002.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.

[NF94] Makoto Namiki and Komei Fukuda. Unfolding 3-dimensional convex polytopes: A package for mathematica 1.2 or 2.0, 1994. `ftp://ftp.ifor.math.ethz.ch/pub/fukuda/mathematica/UnfoldPolytope.tar.Z`.

[NGH04] Xinlai Ni, Michael Garland, and John C. Hart. Fair morse functions for extracting the topological structure of a surface mesh. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 613–622, New York, NY, USA, 2004. ACM.

[Nil71] Nils Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, Inc, 1971. ISBN 07-046573-8.

[O'R00] Joseph O'Rourke. Folding and unfolding in computational geometry. In *Proc. 1998 Japan Conf. Discrete Comput. Geom.*, volume 1763 of *Lecture Notes Comput. Sci.*, pages 258–266. Springer-Verlag, 2000.

[O'R07] Joseph O'Rourke. Band unfoldings and prismatoids: A counterexample. Technical Report 087, Smith College, Oct 2007. arXiv:0710.0811v2 [cs.CG].

[OS89] Joseph O'Rourke and Catherine Schevon. On the development of closed convex curves on 3-polytopes. *J. Geom.*, 13:152–157, 1989.

[Pin07] Val Pinciu. On the fewest nets problem for convex polyhedra. In *Proceedings of the 19th Annual Canadian Conference on Computational Geometry (CCCG 2007)*, pages 21–24. Carleton University, Ottawa, Canada, August 2007.

[Pol03] Konrad Polthier. Imaging maths - unfolding polyhedra, 2003. `http://plus.maths.org/issue27/features/mathart/index.html`.

[Rib95] Paulo Ribenboim. *The New Book of Prime Number Records, 3rd edition*. Springer-Verlag, New York, NY, 1995. ISBN 0-387-94457-5. xxiv+541 pp.

[RT75] Robert Read and Robert Tarjan. Bounds on backtrack algorithms for listing cycles, paths and spanning trees. *Networks*, 5:237–252, 1975.

[Sch97] Wolfram Schlickenrieder. *Nets of Polyhedra*. PhD thesis, Technische Universitat Berlin, 1997.

[She75] G.C. Shephard. Convex polytopes with convex nets. *Math. Proc. Camb. Phil. Soc*, pages 389–403, 1975.

[Sin98] David Singer. *Geometry: Plane and Fancy*. Springer, 1998. ISBN 0387983066.

[Smi97] Malcolm Smith. Generating spanning trees. Master's thesis, University of Victoria, 1997.

[SZ67] I.J. Schoenberg and S.K. Zaremba. On cauchy's lemma concerning convex polygons. *Canadian Journal of Mathematics*, 19:1062–1071, 1967.

[Wei99] Eric Weisstein. Mathworld, 1999. `http://mathworld.wolfram.com/`.

[Yam05] Yamaha, Inc. Realistic paper crafts, 2005. `http://www.yamaha-motor.co.jp/global/entertainment/papercraft/`.

Further Reading

[AER05] Lyuba Alboul, Gilberto Echeverria, and Marcos Rodrigues. Discrete curvatures and gauss maps for polyhedral surfaces. In *Proceedings of the 21st European Workshop on Computational Geometry (EWCG), Eindhoven, Netherlands*, pages 69–72, 2005.

[Alb02] Lyuba Alboul. Curvature criteria in surface reconstruction. *Communications on Applied Mathematics*, pages 7–25, 2002.

[AMOT90] Ravindra K. Ahuja, Kurt Mehlhorn, James Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. *J. ACM*, 37(2):213–223, 1990.

[Bar07] Jonathan F. Bard. Cycles in an undirected graph, 2007. `https://www.me.utexas.edu/~bard/IP/Handouts/cycles.pdf`.

[BCM03] V. Borrelli, F. Cazals, and J.M. Morvan. On the angular defect of triangulations and the pointwise approximation of curvatures. *Comput. Aided Geom. Des.*, 20(6):319–341, 2003.

[BDD+98] Therese Biedl, Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Joseph O'Rourke, Mark Overmars, Steve Robbins, and Sue Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proc. 10th Canad. Conf. Comput. Geom.*, pages 70–71, 1998.

[BDEK99] Marshall Bern, Erik Demaine, David Eppstein, and Eric Kuo. Ununfoldable polyhedra. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, pages 13–16. CCCG'99, Vancouver, British Columbia, Canada, August 1999.

[Ber58] J. Bertrand. Note sur la théorie des polyèdres réguliers. *Comptes rendus des séances de l'Académie des Sciences*, 46:79–82, 1858.

[BG04] Therese Biedl and Burkay Genc. When can a graph form an orthogonal polyhedron? In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG 2004)*, pages 53–56, 2004.

[BLS05] Therese Biedl, Anna Lubiw, and Julie Sun. When can a net fold to a polyhedron? *Comput. Geom. Theory Appl.*, 31(3):207–218, 2005.

[BM04] Devin J. Balkcom and Matthew T. Mason. Introducing robotic origami folding. In *IEEE International Conference on Robotics and Automation*, pages 3245–3250, 2004.

[CGR94] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[CH96] J. Chen and Y. Han. Shortest paths on a polyhedron; part i: computing shortest paths. *Int. J. Comput. Geom. and Appl.*, 6(2):127–144, 1996.

[Cha07] Matthew Chadwick. Celeriac.net - Unfolder, 2007. `http://celeriac.net/unfolder/`.

[CLL+99] H.Y. Chen, I.K. Lee, S. Leopoldseder, H. Pottmann, T. Randrup, and J. Wallner. On surface approximation using developable surfaces. *Graphical models and image processing: GMIP*, 61(2):110–124, 1999.

[DBG+05] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C Hart. Quadrangulating a mesh using laplacian eigenvectors. Technical report, University of Illinois, June 2005.

[DDL00] Erik D. Demaine, Martin L. Demaine, and Anna Lubiw. Folding and cutting paper. In *JCDCG '98: Revised Papers from the Japanese Conference on Discrete and Computational Geometry*, pages 104–118, London, UK, 2000. Springer-Verlag. ISBN 3-540-67181-1.

[DDLO02] Erik D. Demaine, Martin L. Demaine, Anna Lubiw, and Joseph O'Rourke. Enumerating foldings and unfoldings between polygons and polytopes. *Graphs and Combin.*, 18(1):93–104, 2002.

[DDM00] 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.

[DHKL01] Nira Dyn, Kai Hormann, Sun-Jeong Kim, and David Levin. Optimizing 3d triangulations using discrete curvature analysis. In *Mathematical Methods for Curves and Surfaces: Oslo 2000*, pages 135–146, Nashville, TN, USA, 2001. Vanderbilt University. ISBN 0-8265-1378-6.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5.

[DO03] Erik D. Demaine and Joseph O'Rourke. Open problems from CCCG 2002 [problem 43]. In *Proc. 15th Canad. Conf. Comput. Geom.*, pages 178–181, 2003.

[Ede01] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001. ISBN 0-521-79309-2. 162 pp.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5 (6):345, 1962.

[GE04] Chris Gray and William Evans. Optimistic shortest paths on uncertain terrains. In *Proceedings of the 16th Canadian Conference on Computational Geometry, CCCG 2004, Concordia University, Montreal, Quebec, Canada, August 9-11*, pages 68–71, 2004.

[GG04] Tim Gatzke and Cindy Grimm. Improved curvature estimation on triangular meshes. Technical Report 2004-9, Washington University in St. Louis, 2004.

[GMS04] R. Graham, H. McCabe, and S. Sheridan. Pathfinding in computer games. *ITB Journal*, pages 1–31, 2004.

[Gru02] Branko Grunbaum. No-net polyhedra. *Geombinatorics*, 11:111–114, 2002.

[HA03] F. Hétroy and D. Attali. Detection of constrictions on closed polyhedral surfaces. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 67–74, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-698-6.

[Har99] J. Hart. Using the cw-complex to represent the topological structure of implicit surfaces and solids. *Proc. Implicit Surfaces '99, Eurographics/SIGGRAPH*, pages 107–112, 1999.

[HP04a] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum*, 23(3):391–400, September 2004. Proc. Eurographics 2004.

[Hüü06] A. Van Hüüksloot. Papyromania, 2006. `http://www.papyromania.nl/`.

[KO00] Biliana Kaneva and Joseph O'Rourke. An implementation of Chen & Han's shortest paths algorithm. In *Proc. 12th Canad. Conf. Comput. Geom.*, pages 139–146, August 2000.

[Lav06] J. M. Lavric. Litio - 3d, 2006. `http://www.litio3d.com.ar/`.

[Led67] Joshua Lederberg. Hamilton circuits of convex trivalent polyhedra (up to 18 vertices). *The American Mathematical Monthly*, 74(5): 522–527, May 1967.

[LO96] Anna Lubiw and Joseph O'Rourke. When can a polygon fold to a polytope? Technical Report 048, Dept. Comput. Sci., Smith College, June 1996. Presented at *Amer. Math. Soc.* Conf., 5 Oct. 1996.

[Luc06a] Brendan Lucier. Local overlaps in special unfoldings of convex polyhedra. In *Proceedings of the 18th Canadian Conference on Computational Geometry (CCCG 2006)*, pages 97–100, 2006.

[MHC05] J. McCartney, B. K. Hinds, and K. W. Chong. Pattern flattening for orthotropic materials. *Computer-Aided Design*, 37(6):631–644, 2005.

[MM04] P. Morin and J. Morrison. The geometry of carpentry and joinery. *Discrete Applied Mathematics*, 144(3):374–380, 2004.

[MS04] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 259–263, New York, NY, USA, 2004. ACM.

[MW00] D. S. Meek and D. J. Walton. On surface normal and gaussian curvature approximations given data sampled from a smooth surface. *Comput. Aided Geom. Des.*, 17(6):521–543, 2000.

[O'R99] Joseph O'Rourke. Computational geometry column 35. *Internat. J. Comput. Geom. Appl.*, 4–5:513–515, 1999. Also in *SIGACT News*, 30(2):31-32 (1999), Issue 111.

[OS89a] Joseph O'Rourke and Catherine Schevon. Computing the geodesic diameter of a 3-polytope. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 370–379, New York, NY, USA, 1989. ACM. ISBN 0-89791-318-3.

[Pol02] Konrad Polthier. Polyhedral surfaces of constant mean curvature (extract: chapters 1,3,4), 2002.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PW98] H. Pottmann and J. Wallner. Approximation algorithms for developable surfaces - technical report no 51. Technical report, Institut fur Geometrie, April 1998.

[Rev02] Revcad Software, Inc. Sheet lightning, 2002. `http://www.revcad.com/Sheet5/index.html`.

[SBBC00] Alla Sheffer, Michel Bercovier, Ted D. Blacker, and Jan Clements. Virtual topology operators for meshing. *International Journal of Computational Geometry and Applications*, 10(3):309–331, 2000.

[SH02] Alla Sheffer and John Hart. Seamster: Inconspicuous low-distortion texture seam layout. *Proc. IEEE Visualization*, pages 291–298, 2002.

[SMS+03] Tatiana Surazhsky, Evgeni Magid, Octavian Soldea, Gershon Elber, and Ehud Rivlin. A comparison of gaussian and mean curvatures estimation methods on triangular meshes. In *Proceedings of the*

2003 IEEE International Conference on Robotics and Automation, ICRA 2003, September 14-19, 2003, Taipei, Taiwan, pages 1021–1026. IEEE, IEEE, 2003.

[STL06] Idan Shatz, Ayellet Tal, and George Leifman. Paper craft models from meshes. *Vis. Comput.*, 22(9):825–834, 2006.

[Tam02] Tama Software, Inc. Pepakura, 2002. `http://www.tamasoft.co.jp/pepakura-en/`.

[Tau95] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 902, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7042-8.

[TLT04] K. Tai, W. Liu, and G. Thimm. Unfolding and flat layout design of non-manifold 3d folded structures. *Computer-Aided Design & Applications*, 1(1-4):439–448, 2004.

[VA97] Kasturi R. Varadarajan and Pankaj K. Agarwal. Approximating shortest paths on a nonconvex polyhedron. In *IEEE Symposium on Foundations of Computer Science*, pages 182–191, 1997.