Folding Orthogonal Polyhedra

by

Julie Sun

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1999

Abstract

In this thesis, we study foldings of orthogonal polygons into orthogonal polyhedra. The particular problem examined here is whether a paper cutout of an orthogonal polygon with fold lines indicated folds up into a simple orthogonal polyhedron. The folds are orthogonal and the direction of the fold (upward or downward) is also given. We present a polynomial time algorithm to solve this problem.

Next we consider the same problem with the exception that the direction of the folds are not given. We prove that this problem is NP-complete.

Once it has been determined that a polygon does fold into a polyhedron, we consider some restrictions on the actual folding process, modelling the case when the polyhedron is constructed from a stiff material such as sheet metal. We show an example of a polygon that cannot be folded into a polyhedron if folds can only be executed one at a time. Removing this restriction, we show another polygon that cannot be folded into a polyhedron using rigid material.

Acknowledgements

The results of this thesis arise from joint research with my supervisor, Anna Lubiw. I thank Anna for her guidance and support since the very beginning.

I would also like to thank my readers Therese Biedl and Paul Kearney for their valuable comments and suggestions. Extra thanks go out to Therese for her ideas in the construction of the polyhedron in Section 3.2.

Contents

List of Figures

Chapter 1

Introduction

Imagine you have a simple cube-shaped gift and want to make a cardboard box to put it in. You have all the materials to make it (i.e. cardboard, scissors, and tape), and all you need to do is figure out what shape to cut from the cardboard that will form a box. You want to cut out a single piece that you can fold together and tape at the edges. The most common shape people will generally come up with is the cross or T-shaped figure composed of six squares. Now imagine your birthday is coming up, and you know your mother always makes her own gift boxes shaped exactly to fit your presents and you find a bunch of cardboard cutouts, one of which will be used to hold your present. For each of these cutouts, you can try folding them up to see which one actually forms a closed shape. And of these, which one looks to fit the gift you had wished for. The former problem deals with determining an unfolding of a shape. The latter problem of determining whether a cardboard cutout folds to form a closed shape is the essence of this thesis.

This thesis is organized as follows. The first section considers the problem of determining whether an orthogonal polygon with folds indicated, folds up into an orthogonal polyhedron when the directions of the folds are given. An algorithm for solving this is given.

The second section focuses on the same problem but with the added difficulty that the directions of the folds are not given (i.e. indicated folds can either be mountain or valley folds). We show that this problem with undirected folds is NP-complete.

The last section considers the problem when the cutout is made of some rigid material like sheet metal. With a rigid material, faces cannot give way to allow the movement of other faces of a cutout. This is contrary to folding with a supple material such as plain white paper. We explore some polygons that cannot be folded with this restriction.

Before proceeding, we present some related works that have been done in the area. Then a precise definition of an orthogonal polyhedron is given, along with terminology used throughout the thesis.

1.1 Background and Motivation

The problem of folding polygons appears in such applications such as origami and packaging. Origami is the Japanese art of paper folding. In packaging, the process of making a box requires folding some pre-cut material of some polygonal shape into a closed box [RW91]. Unfolding problems may also appear in these areas as well as in packaging problems. More research has been done on unfolding problems than on folding problems, simply because it is more practical to begin with an object and determine a polygon that forms it. In practice, it is not usually the case where you have some arbitrary polygon and want to fold it into some polyhedral shape. We describe two papers which are particularly related to the topic of this thesis. The first paper, by Lubiw and O'Rourke [LO96], discusses the problem of folding a polygon into a polyhedron. They give an $O(n^2)$ algorithm, for a polygon of n vertices, which determines whether the given polygon folds to a polyhedron. The algorithm is based on a theorem by Aleksandrov that says that if the edges of a polygon can be matched to form a surface homeomorphic to a sphere in which the complete angle at each vertex is no more that 2π, then the polygon does fold to a polyhedron [Ale58].

In the second paper, Biedl *et al.* write about unfolding some classes of orthogonal polyhedra [BDD$^+$98]. In particular, they study two classes of polyhedra which they call *orthotubes* and *orthostacks.* In a paper by Demaine *et al.* [DDL$^+$99], the authors explore how a given polygon can fold into several different polyhedra with differing crease patterns in the polygon. In particular, they show how the Latin cross can fold into four different shapes in addition to the cube.

The main question has been "Does every simple polyhedron unfold, by cutting along its edges, to a simple polygon that does not self-intersect?"

There has not been much work done in the area of folding polyhedra, and much remains to be discovered. If we are given a polygon with directed folds, there is certainly a unique orthogonal folding. For non-orthogonal foldings, there may be several foldings. If the directions of the folds are not given, a polygon may have different foldings that can result [LO96].

The last section of this thesis considers the problem of determining the sequence of folds to execute that will result in the folded polyhedron if the cutout is made of sheet metal. Most foldings can be realized, but it is not always trivial which bends to execute first since some bends may make other bends impossible. Some research has been done in this area of bending sheet metal. This problem is more of a planning problem in which the goal is to figure out a valid series of steps to achieve the desired folded shape. Gupta *et al.* present a planning process system for a sheet metal bending press-brake [GBKK98]. The system takes the design for some part that is to be constructed from a flat piece of sheet metal. In particular, the design indicates the folds to be made in order to make the part. Given this design, the system will determine a plan for folding the cutout into the desired part. The system also controls the press-brake (the machine that makes the bends).

Of theoretical interest, a similar problem to the sheet metal bending problem is a two dimensional problem. This type of problem is called the polygonal chain folding and unfolding problem [BDD$^+$99] (or the carpenter's ruler problem). Instead of folding flat faces, the polygonal chain folding problems attempts to fold or unfold solid line segments without allowing them to intersect. This problem has applications in the movement of robotic arms. A robotic arm is represented as line segments with joints. The movements of these segments depict the movements a robotic arm can do. A chain may either be closed or open. For open chains, the goal is to straighten the chain and for closed chains, the goal is to convexify the chain.

One of the obvious reasons for studying foldings and unfoldings of polyhedra is for model making. It is often said that one can appreciate and better understand polyhedra by making models of them. These models can be used to study molecular geometries [HH88]. Although the ball and stick method is normally used for modelling atoms and molecules, cardboard models can also be useful. They may be easier to combine together to form other polyhedra, and sometimes more helpful in *space filling* problems.

1.2 Definition

Before we start with the orthogonal polyhedron folding problem, it is necessary to establish a clear definition of *orthogonal polyhedron.* This is not a simple task. Cromwell gives a history of the attempts made to define polyhedra [Cro97]. He concludes the discussion with a refined definition of his own which still seems vague and incomplete. In fact, many resources [Cro97, Cox63, Sen88] give similar definitions of polyhedra which still differ in some respects.

In this thesis, we are looking at a particular class of polyhedra – orthogonal polyhedra – whose components (vertices, edges, and faces) lie parallel to the x, y, and z axes. Since the definitions in the literature describe non-orthogonal polyhedra, it will be necessary to define our own interpretation of what an orthogonal polyhedron is. Clearly our polyhedra are nonconvex since the only possible convex orthogonal polyhedra are rectangular prisms and cubes. Senechal [Sen88] encourages the reader to define polyhedra depending on the properties being studied. The reader should look at the kinds of polyhedra one is willing to accept such as *star polyhedra, toroidal polyhedra, infinite polyhedra*, or polyhedra whose faces are *skew polygons*. From there, the reader can decide on an appropriate definition to work with.

We first state the definition that will be used, and follow with a discusion of how we arrived at it based on a definition for general polyhedra by Coxeter [Cox63]. The exact meaning of the definition is given as well as some examples of shapes we consider to be orthogonal polyhedra and some which we do not.

> An *orthogonal polyhedron* is a finite connected set of plane orthogonal polygons called faces, with the properties that
>
> (1) if two faces intersect, it is only at a common vertex or a common edge, and never in an interior point of a face,
>
> (2) every edge of every face is an edge of exactly one other face, and
>
> (3) the faces surrounding each vertex form a single circuit.

In this definition, *orthogonal polygons* simply means the edges of the polygon are parallel to the coordinate axes.

This definition was mostly derived from a definition of *polyhedron* given by Coxeter [Cox63] which is restated below.

> A polyhedron may be defined as a finite, connected set of plane polygons, such that every side of each polygon belongs also to just one other polygon, with

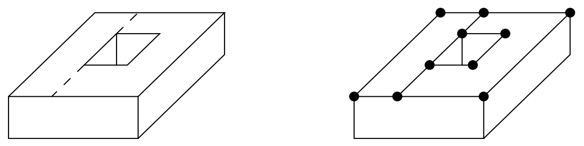
> the proviso that the polygons surrounding each vertex form a single circuit (to exclude anomalies such as two pyramids with a common apex). The polygons are called *faces*, and their sides *edges*. ...we insist that the faces do not cross one another. Thus the polyhedron forms a single closed surface, and decomposes space into two regions, one of which, called the *interior*, is finite.

In his definition, *sides* is synonymous to *edges* as used in our definition. Coxeter also defines *plane polygon* which is restated below.

> We define a p-gon as a circuit of p line-segments $A_1A_2, A_2A_3, \ldots, A_pA_1$, joining consecutive pairs of p points $A_1, A_2, \ldots, A_p$. The segments and points are called *sides* and *vertices*. ...we shall insist that the sides do not cross one another. If the vertices are all coplanar we speak of a *plane* polygon, otherwise a *skew* polygon. A plane polygon decomposes its plane into two regions, one of which, called the *interior*, is finite. We shall often find it convenient to regard the p-gon as consisting of its interior as well as its sides and vertices. We can then re-define it as a simply connected (i.e. no holes) region bounded by p distinct segments.

Coxeter's definition captures some key ideas of a polyhedron that some other definitions don't. For one, it states the finiteness of the polyhedron which some definitions do not [Grü77]. Coxeter is also explicit in saying the faces must be plane polygons as opposed to skew polygons. The part where he states that space is decomposed into two regions prevents objects with hollow spaces inside it from being allowed by the definition, these would decompose space into three regions, two of which would be considered the *exterior*.

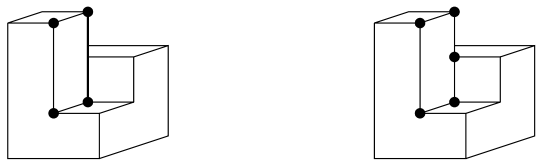
The second condition of our definition requires that every edge of every face belong to exactly one other face. This condition is two-fold. If an edge of a face does not belong to any other face, then the polyhedron is not *closed*. If an edge belongs to more than two faces, then the polyhedron decomposes space into more than two regions. So we require that exactly two faces share an edge. Here, it is important to say exactly what a face is and what a polygon is. In particular, the definition does not exclude two coplanar faces sharing an edge. Thus, for example, the polyhedron shown in Figure 1.1 is valid. Notice that this polyhedron contains faces (the top and bottom faces) with holes in them, which Coxeter clearly prohibits. Thus we need to decompose the top face into two faces, as shown on the right of Figure 1.1.



The top face can be composed of two adjacent faces, one of which has had two vertices added in order for edges to match one-to-one.

Figure 1.1: Inserting intermediary vertices

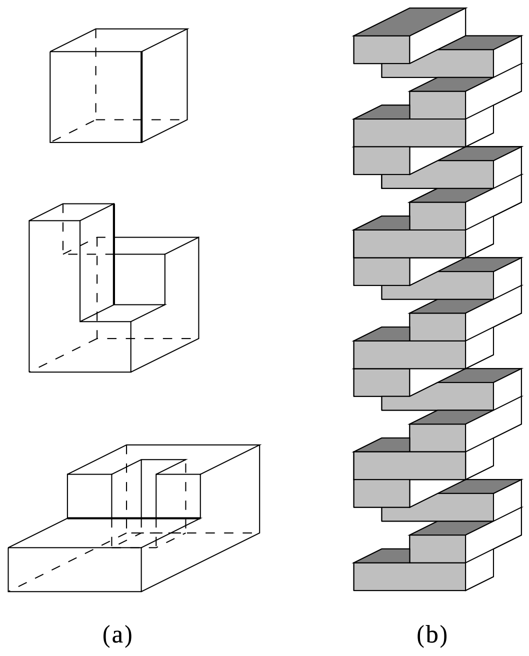
It is also important to note that a polygon may have two adjacent collinear edges, though our pictures of polyhedra often do not make this explicit. Thus, for example, the polyhedron in Figure 1.2 is valid, but we must insert an intermediary vertex along an edge in order for the edge matching condition in the definition to hold.



The bold line shows an edge shared by three faces; by adding a vertex on that line we create an extra edge so that every edge is shared by exactly two faces.

Figure 1.2: Inserting intermediary vertices

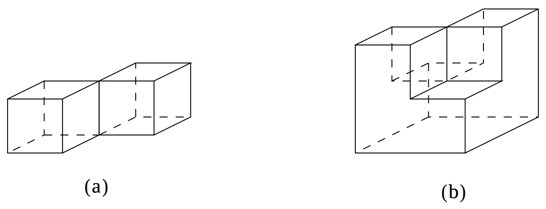
Likewise for the polyhedra shown in Figure 1.3. Some edges of a polygon are shared by multiple polygons, but are all valid polyhedra.



(a) three polyhedra with an edge (shown with a thick line) shared by multiple polygons
(b) a spiralling polyhedron with many edges, each shared by three polygons

Figure 1.3: Some polyhedra with edges shared by many faces

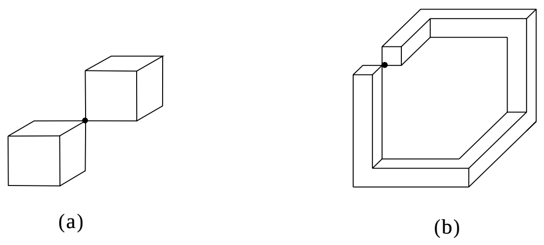
Figure 1.4 shows some examples we wish to exclude from our definition. These polyhedra will not satisfy the definition in either one of two ways. If the polyhedron is interpreted to have four faces which meet at the centered edge, this violates the necessity that every edge is an edge of just one other face. The other way to view the polyhedron is such that two of the coplanar faces is actually a single face in the original cutout which creates an intersection among faces. This too will violate one of Coxeter's conditions.



(a) shows two cubes joined at a vertex; (b) another shape we do not wish to consider as a polyhedron

Figure 1.4: Examples of shapes that are not polyhedra

The last condition of our definition requires there to be only one circuit of polygons at any vertex. This rules out any shapes like the ones illustrated in Figure 1.5.



(a) shows two cubes joining at an apex; (b) shows another shape we do not wish to consider as a polyhedron

Figure 1.5: Examples of shapes that are not polyhedra

A more precise description of a circuit by Cromwell [Cro97] is restated below.

> *Let V be any vertex and let $F_1, F_2, ..., F_n$ be the n polygons which meet at V. It is possible to travel over the polygons F_i from one to another without passing through V.*

Of the two examples shown in Figure 1.5, one had two cubes joined at a common vertex. The other shape is one continuous shape whose ends meet at a common vertex. Instances like the first example will not arise in the problem examined in this thesis since the input to the problem is a connected graph of faces – the polygon – and no input will yield such a shape. However, instances of the second example can arise from a valid input.

In some definitions of polyhedra, two faces may only meet at one common edge. However, we wish to accept polyhedra such as the ones in Figure 1.6 in which some faces may join at several common edges.

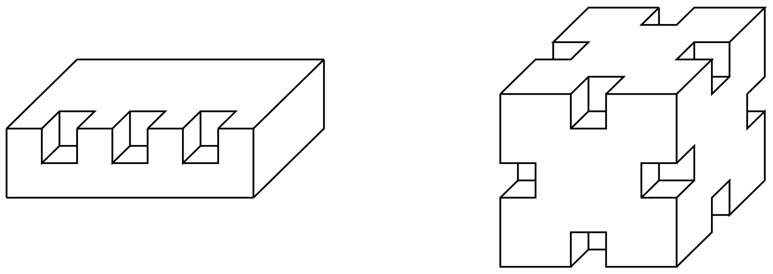


Figure 1.6: These are valid orthogonal polyhedra whose faces meet another face at one or more edges

1.3 Terminology

Before going on, we present some terminology used throughout this thesis. Some terms are depicted in the diagram of a simple cutout shown in Figure 1.7.

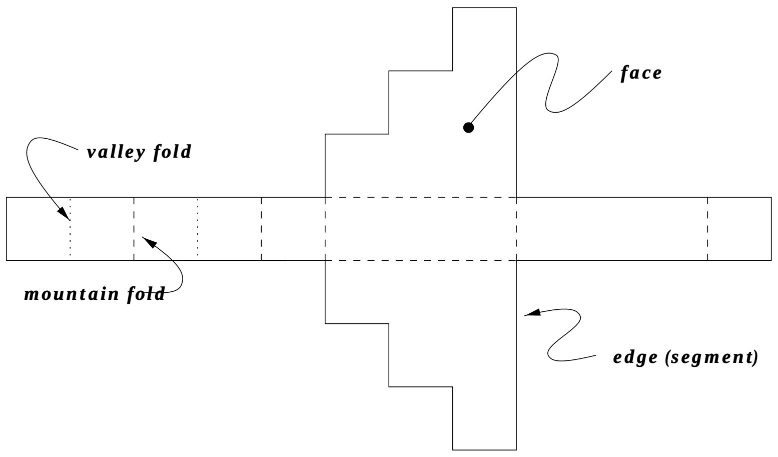


Figure 1.7: Some terminology used in this paper

Mountain Fold. A mountain fold means that the *interior* of the faces sharing a fold are folded towards each other.

Valley Fold. A valley fold means that the *interior* of the faces sharing a fold are folded away from each other.

Fold Line. A fold line is shown in the diagram as either dashed or dotted lines. Dashed lines indicate mountain folds, and dotted lines indicate valley folds. All folds are orthogonal, meaning that they should bend at a right angle.

Face. A *polygonal face* is a sub-polygon of the given orthogonal polygon. Polygonal faces are bounded by fold lines and possibly edges of the cutout. A *polyhedral face* is either a single polygonal face or the union of adjacent co-planar polygonal faces.

Polyhedral Net. An unfolding of a polyhedron is called a net. A net shows both the outline of the unfolding and the fold lines. Directions of the folds are not necessarily shown.

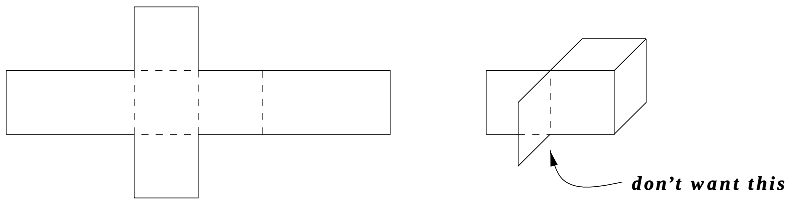
Chapter 2

The Polyhedron Folding Problem

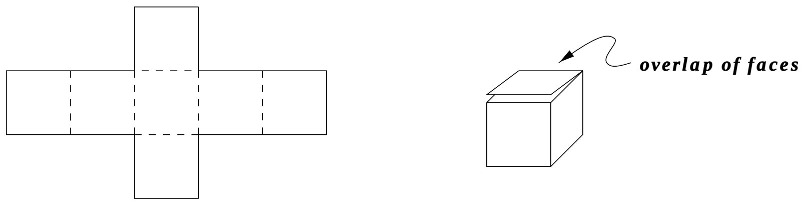
Suppose we are given a paper cutout where the outline is an orthogonal polygon with no holes or cuts in it, and with orthogonal fold lines indicating the direction of the fold. The directions are mountain folds (folding down) and valley folds (folding up). And every fold line must fold orthogonally. An orthogonal fold is one in which the dihedral angle between two faces is exactly 90 degrees. Each fold line is orthogonal and must start and end on a point lying on the boundary of the polygon, and no two fold lines may cross, although they may meet at a vertex of the polygon. Given such a polygon, we wish to determine if the polygon folds into a simple orthogonal polyhedron, as defined in Chapter 1. Faces may not overlap or intersect. See Figure 2.1 for some examples.

The basic idea to determine if a polygon folds into a valid polyhedron, is simple: fold the polygon according to the fold lines given, and check that the folded object forms an orthogonal polyhedron. Two main steps are required to do this:

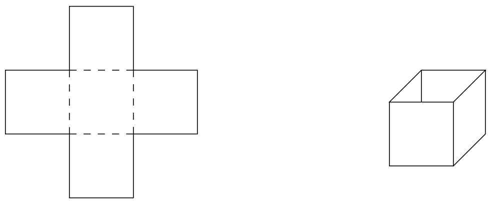
1. Compute the (x, y, z) coordinates of each vertex when the polygon is completely folded.
2. Check that the conditions of the definition of an orthogonal polyhedron hold. *Note:* We need not check the connectedness of the polyhedron since our problem is given



(a) an example where a paper cutout folds into a box with intersecting faces



(b) an example where a paper cutout folds into a box with an overlapping "lid"



(c) an example where the cutout does not fold into a closed box

Figure 2.1: Three examples that do not satisfy the requirements

a cutout in which the faces are connected. This also ensures all our faces are plane polygons since all folds are orthogonal.

In Section 2.1, we will describe an algorithm for step 1 that runs in linear time. In Section 2.2 we will describe an alogrithm for step 2 that runs in quadratic time. This is not the most efficient possible, so in Section 2.3 we briefly mention a space-sweep technique that will improve the efficiency of the algorithm for step 2.

2.1 Computing 3D Coordinates

The input to this problem is a sequence of points, as (x, y) coordinates, defining an orthogonal polygonal cutout. Pairs of points are also given to represent fold lines each of which is either a mountain fold, or a valley fold. To determine whether this cutout folds to form a polyhedron, it is necessary to check some properties of the folded polyhedron. To check these properties requires the coordinates of the vertices, edges, and faces when they are in their folded positions in three dimensional space. Assume that the polygon is initially lying in the xy plane. Taking the dual of the polygonal net as shown in Figure 2.2 (this is done by assigning a node to each polygonal face and joining two nodes if their respective faces share a fold) we can process every face by traversing the dual. We start with a leaf node and traverse the dual in a depth first manner until all coordinates of every polygonal face have been computed.

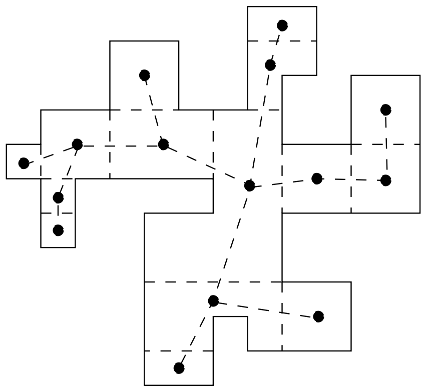

An orthogonal polygon and its dual,
which is used to process faces of the polygon.

Figure 2.2: A method to traverse the faces

To compute the 3D coordinates, we must colour the surface of the polygon black (the side which the z^+ axis points out from), and the underside of the polygon (which the z^- axis points out from) white. To determine the coordinates, there are some attributes that need to be stored for each face and each fold line between two faces:

Face plane — the plane in which the folded face lies (e.g. parallel to xy, yz, or xz).

Fold axis — the axis in which a fold lies parallel to (i.e. x, y, or z). This is determined simply by the coordinates of the fold line – e.g. If a fold has coordinates (3,5,1) and (3,5,5), then we know the fold axis is z.

Fold type — either mountain or valley as given by the input.

Face orientation — the direction that comes out of the black side of a face (i.e. x^+, x^-, y^+, y^-, z^+, or z^-); this is needed to determine the coordinates of a face. Suppose the current face being processed is connected by a mountain x-fold from the previous face processed, which lay in the xy plane. Without knowing which side of the previous face is coloured black, we cannot know which direction the current face will fold to, relative to the previous face. This is illustrated in Figure 2.3.

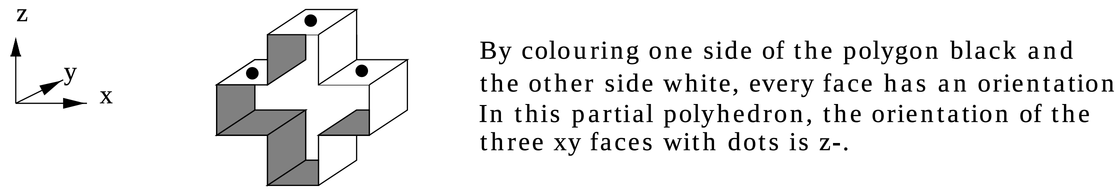
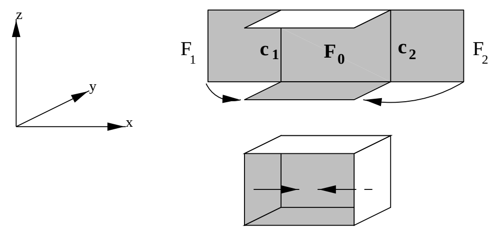

z
y
x
By colouring one side of the polygon black and the other side white, every face has an orientation. In this partial polyhedron, the orientation of the three xy faces with dots is z-.

Figure 2.3: The face orientation

Fold position — the position of a fold on a face is needed to determine the orientation of the faces it is joined to. The example in Figure 2.4 shows a vertical fold, c_1, to the right of the centre face, F_0 and another to the left, c_2. So a vertical fold to the right of a face means the black side will point to the left, while a fold to the left of the centre face means the black side of the next face will point to the right. Consider fold c_1. It is in the x^- direction of F_0 (i.e. to the left) and F_1, which folds towards the y^- axis, will have its black side facing in the x^+ direction. Now consider c_2. Its position is x^+ and folding F_2 towards the y^- axis means its orientation will be x^-. Hence the position of a fold is necessary in order to determine the orientation of a proceeding face processed. When the vertices of a polygon are given, it is standard practice to list the vertices in such a way that you traverse the polygon counter-clockwise. By doing so, the interior of the face is to the left. Using this convention, we can determine the position of a fold line with respect to the face it lies on.



The position of a fold on a face is needed to determine the orientation of the faces it is joined to. In this example a vertical fold to right of the centre face means the black side will point to the left, while a fold to the left of the centre face means the black side of the next face will point to the right.

Figure 2.4: The fold position

Here is an outline of the algorithm to compute 3D coordinates:

1. Let $F_1, F_2, ..., F_k$ be the faces in the polygonal cutout in the depth-first order they are traversed.

2. Place F_1 arbitrarily in the xy-plane .

3. For i=2, ..., k,

   (a) Let F_j be the parent of F_i.

   (b) Let c_i be the fold between F_i and F_j.

   (c) Determine the face plane of F_i.

   (d) Determine the fold axis and fold position of c_i.

   (e) Determine the face orientation of F_i from the fold type and the fold position of c_i.

   (f) Calculate the 3D coordinates of F_i.

This method of determining the (x, y, z) coordinates takes $O(n)$ time, where n is the total number of vertices among the faces. Since we are travelling along the faces in a depth first manner, every vertex will be processed a constant number of times.

2.2 Condition Verification

The problem handled in this section is to check the conditions (1), (2), and (3), of the definition of an orthogonal polyhedron (p.5). The input to the problem is the 3D coordinates of the faces as computed in Section 2.1. The output will be some representation of the folded polyhedron if the input does indeed fold into one.

Edelsbrunner describes an *incidence graph* as a data structure for a convex polyhedron [Ede87]. This graph can also be used for orthogonal, non-convex polyhedra. An incidence graph is a hierarchical graph representing each of the components of a polyhedron – vertices, edges, and faces. The empty space and the polyhedron itself may also be represented in the graph. The first level of the graph is a row of nodes, representing the faces of a polyhedron. The next row of nodes represents the edges of the polyhedron, while the last row represents the vertices of a polyhedron. Links (edges of the incidence graph, not to be confused with edges of the polyhedron) occur between nodes in adjacent rows and represent containment relationships. That is, if a face, f_1 contains four edges, e_1, e_2, e_3, and e_4, then there exists a link between the node representing f_1 and each of the nodes representing e_1 to e_4. Similarly, since every edge is defined by two vertices and is shared by two faces, every node representing an edge is linked to exactly two nodes representing vertices and two nodes representing faces. Each vertex will be connected to a cyclic list of edges that originate from it. Similarly, each face will be connected to a cyclic list of edges around it. The incidence graph for an L-shaped block is shown in Figure 2.6.

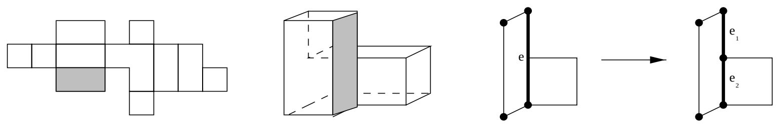


Figure 2.5: When two faces meet at a part of an edge

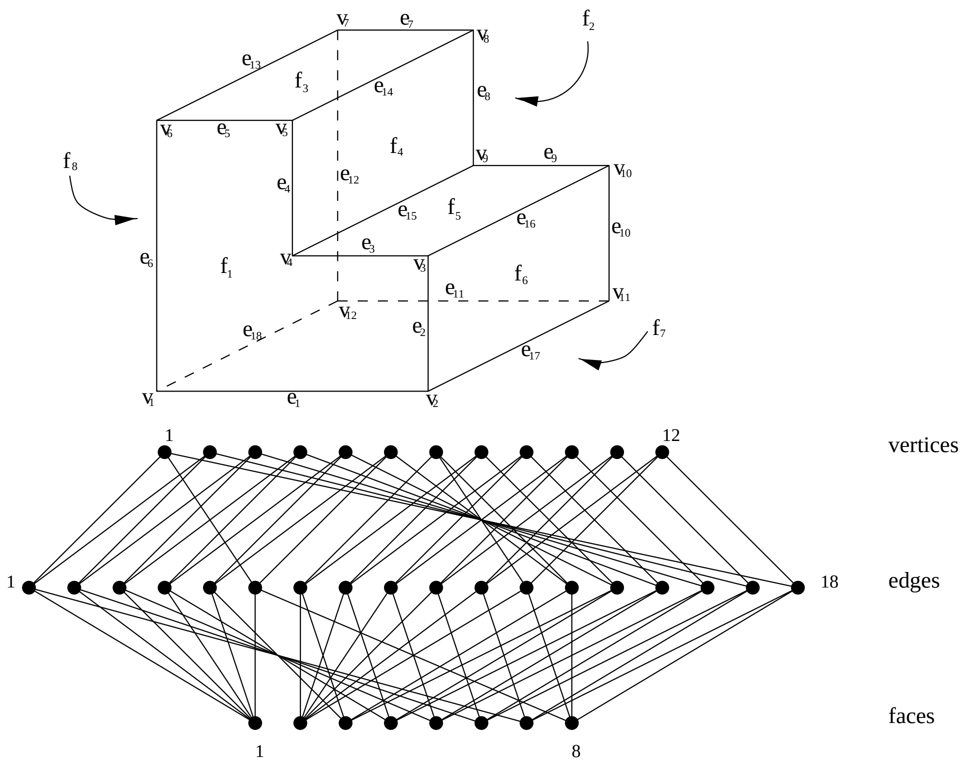
Recall that the faces of the polyhedron are not the same as the faces extracted from the 2D folding. In Figure 2.5 the shaded face is defined by four edges of the polygonal cutout. However, for the edge-sharing condition of the definition to hold, an extra vertex must be added to the face. Otherwise the edges will not be matched. Hence the edge e of the face is replaced with two edges, e_1 and e_2. Let this modified face be called the *upgraded face.*

A brute force algorithm for checking the three conditions follows.

1. **Compute the intersection between every pair of faces**

   In this step we test the first condition of our definition: *If two faces intersect, it is only at a common vertex or a common edge, and never in an interior point of a face*, and we build the incidence graph. If the intersection is a line segment in the interior of any one of the faces (one face penetrates the other) or if it is a polygon (they overlap), then HALT. If the intersection is empty or a vertex, then do nothing. If the intersection of the two faces is an edge of both faces, add that edge to the incidence graph. If the intersection is a segment (or segments) of an edge, then split the edge by adding sufficiently many extra vertices. Record the new edges of the upgraded face and delete the previous one. There are at most $O(m)$ vertices that can possibly be added to a face since a vertex is added only if there is an incident edge at that point; since there are m edges in the polyhedron.

   At the beginning of this step, the incidence graph is empty. For each face being processed, we add a node to the incidence graph representing a face of the polyhedron. Each time we add an edge to the incidence graph, we must add both a node representing



Cyclic order of edges at each vertex

v_1	e_1	e_6	e_{18}
v_2	e_1	e_{17}	e_2
v_3	e_2	e_{16}	e_3
v_4	e_3	e_{15}	e_4
v_5	e_4	e_{14}	e_5
v_6	e_5	e_{13}	e_6
v_7	e_7	e_{12}	e_{13}
v_8	e_8	e_7	e_{14}
v_9	e_9	e_8	e_{15}
v_{10}	e_{10}	e_9	e_{16}
v_{11}	e_{11}	e_{10}	e_{17}
v_{12}	e_{12}	e_{11}	e_{18}

Counter-clockwise cyclic order of edges for each face

f_1	e_1	e_2	e_3	e_4	e_5	e_6
f_2	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
f_3	e_5	e_{14}	e_7	e_{13}		
f_4	e_4	e_{15}	e_8	e_{14}		
f_5	e_3	e_{16}	e_9	e_{15}		
f_6	e_2	e_{17}	e_{10}	e_{16}		
f_7	e_1	e_{18}	e_{11}	e_{17}		
f_8	e_6	e_{13}	e_{12}	e_{18}		

Figure 2.6: The incidence graph for the L-shaped block

the edge and one or two nodes representing the vertices of that edge to the incidence graph. If a node representing a vertex already exists, we use the existing node. Links must also be added to join the face to the newly added edge and one or two links to join the edge to the vertex that is incident to it.

2. **Test whether every edge of every polygon is an edge of exactly one other polygon.**

   For each face (as well as updated faces), check that every edge is an edge of exactly one other face. Using the incidence graph, this is easily done by checking that every edge is linked to exactly two different faces. This condition may also be verified in the previous step. Once a face has been compared to all the other faces, if any edge of that face has not been inserted into the incidence graph, then HALT. That edge does not belong to any other face, and this condition fails.

3. **Test whether the polygons surrounding each vertex form a single circuit.**

   For each vertex, compute and check the cyclic order of edges originating from the vertex. An incidence graph is used to check for the existence of multiple circuits at each vertex of a polyhedron. An upgraded version of the incidence graph imposes a direction on the representation of each edge. By doing this, an edge, $\vec{e}$, has an origin, u, a destination, v, and a *left* face. The same edge with the opposite direction is also represented. For each vertex, choose an edge, incident to that vertex, to begin with. Find the face that is to the left of that edge. Choose an edge of that face whose origin is the vertex being checked. Find the face that is to the left of the opposite edge of the last edge chosen, and repeat as before. When an edge chosen is the same as the first edge that was chosen, stop. Check that there are no other edges that originate from that vertex. If there are, then there are multiple circuits at that vertex and the polygon can be rejected. And so the whole procedure for checking vertices takes $O(m)$ time where m is the number of edges in the polyhedron. If there is more than one circuit, then there will be a pair of edges with the same origin but with no face in common, in which case the test will fail.

Analysis

The first subtask requires $O(f^3 log f)$ time where f is the number of faces since every face is checked against every other face. The second task checks that every edge of every face is an edge of some other face. Using the incidence graph, this task can be accomplished in $O(m)$ time where m is the number of edges in the polyhedron. The last subtask checks for multiple circuits about the vertices of the polyhedron. This can also be checked in $O(m)$ time where m is the number of edges of the polyhedron. Although the method is not necessarily efficient, the algorithm runs in polynomial time.

2.3 Improving the Algorithm

The brute force method of checking every face against every other face can be improved by using a space sweep technique. Our problem is to detect any intersections or the overlapping of two dimensional objects (plane polygons) in three dimensional space. There are many algorithms for detecting such intersections of 2D objects in 2D space – often called *plane-sweep* techniques. There are also algorithms for detecting intersections of 3D objects in 3D space – called *space sweeping.* Dobkin and Kirkpatrick give an $O(log^2 n)$ algorithm for the detection of polyhedral intersections [DK82].

Combining the two methods, it is possible to solve the problem of this chapter in time faster than $O(n^2)$. Details of such an algorithm are not given here.

When two faces overlap, the area that is common to both faces is sometimes called their intersection. Although they are merely touching, they are indeed intersecting in the plane. In two dimensions, the area of intersection between two polygons can be calculated as the boolean AND mask operation. Widmayer and Wood give an algorithm for this operation [WW86] which can be computed in $O(n \log n + p)$ time where n is the number of polygons, and p is the number of contour edges. Contour edges are those that bound the area of intersection.

Chapter 3

The Polyhedron Folding Problem With Undirected Folds

In this section we examine the problem of determining whether an orthogonal polygon folds into an orthogonal polyhedron if the directions of the folds are not given. As before, each fold line is orthogonal and must start and end on a point lying on the edges of the polygon, and no two fold lines may cross, although they may meet at a vertex of the polygon. Every fold is orthogonal, that is the dihedral angle between two faces with a common fold line is 90 degrees. The same restrictions that apply to the problem in the previous section apply again to the polyhedron in this problem. That is, no two faces may intersect or overlap, and the solid formed by the polyhedron must be fully bounded by faces with no open surface. The natural method to solve this problem would be to try a fold in one direction and continue folding the remaining folds of the polygon in the same manner. If it fails, backtrack to the last fold and try folding in the other direction. This yields an exponential time algorithm. Can we do better than exponential? We claim that this problem is NP-complete.

Before examining the 3D problem and showing it is indeed NP-complete, we will examine a 2D version of the problem and show that it is NP-complete. Rather curiously, this will not immediately imply NP-completeness of the 3D version – more work will still be needed.

In the 3D problem, an orthogonal polygonal net is given along with orthogonal folds. The 2D version is simpler. Reducing the 3D problem by one dimension, the problem becomes,

> Given a sequence of straight line segments with joints between the segments, determine if the line segments fold to form a simple orthogonal polygon. The joints must bend, either up or down, at right angles.

Imagine the line segments are joined together, similar to a carpenter's ruler except that the segments are of different lengths. In the 2D problem, the input is a sequence of n positive integers representing the lengths of line segments. The bends at each joint must be orthogonal and no two segments can overlap or intersect. Here are some examples to illustrate configurations which are acceptable and not acceptable orthogonal polygons and their respective sequence of integer line segment lengths.

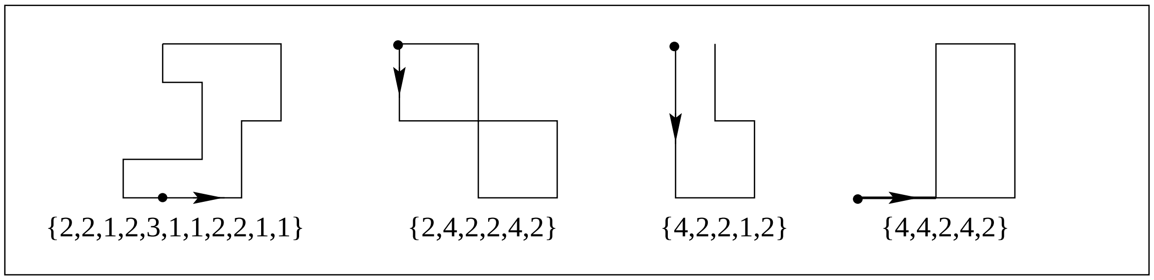


Figure 3.1: The first figure is okay while the rest are not

It turns out this problem is NP-complete. The proof follows.

3.1 NP-completeness of the 2D Problem

Theorem 1: *The 2D folding problem is NP-complete.*

Proof: We first show that the 2D folding problem is in NP. If the input to the problem is a chain of integer-length links, then the evidence used to verify that a closed simple orthogonal polygon can be formed consists of the direction of the "fold" at each joint of the chain.

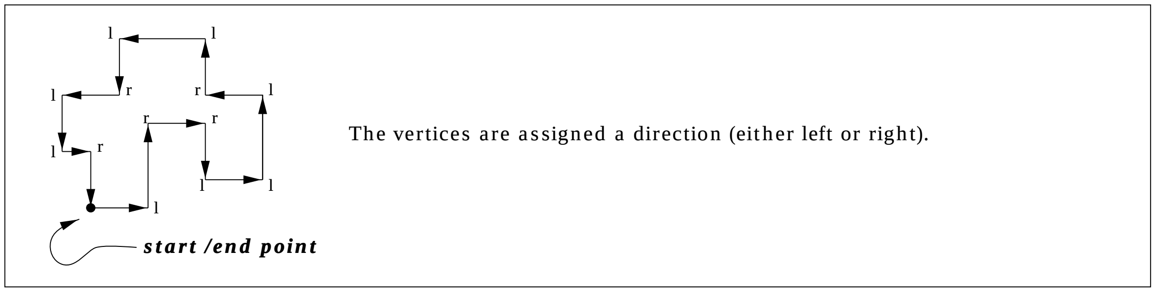

l
l
r
r
l
l
r
r
l
r
l
l
l
start /end point
The vertices are assigned a direction (either left or right).

Figure 3.2: An assignment of directions to vertices

The coordinates of the vertices in the plane can be calculated (from the assigned directions) in polynomial time. Finally the polygon needs to be checked for intersecting and overlapping line segments. This too can be done in polynomial time. Therefore the 2D folding problem is in NP.

To determine if the input instance to the 2D problem yields a simple orthogonal polygon, two necessary conditions must hold. In the final polygon, every other integer length in the input will be a horizontal edge and all others will be a vertical edge or vice versa. The second fact is that all the integers (associated to the edges) are assigned a positive or negative sign. Traversing the polygon, each edge acquires a direction. More precisely, if an edge goes upward or to the right, we label it as a positive edge, otherwise it is a negative edge. Then the sum of these signed integers, within the set of horizontal edges or the vertical edges, must be zero. In other words, for the input to yield a "yes" output, the two sets of integers must have some partition such that the sums are equal. To show that the 2D folding problem is NP-complete, we reduce the Partition Problem to it. The Partition Problem is to determine if a set of positive integers, S, can be partitioned into two subsets, S_1 and S_2, such that the sum of the elements of S_1 is equal to the sum of the elements of S_2, where $S = S_1 \cup S_2$. The Partition Problem is NP-complete [GJ79].

Here is a description of the transformation that converts an input instance to the Partition Problem to an input instance to the 2D folding problem. Suppose $S = \{x_1, x_2, \ldots, x_n\}$ is an instance of the Partition Problem. We want to construct from S an instance S' of the 2D Folding Problem.

The idea is to construct an orthogonal polygon forming a *C-clamp* like structure and adding the elements from S as horizontal edges, and some vertical edges between these edges to join at the ends of the C-clamp. An unfolding of the resulting polygon is S'. An example in Figure 3.3 shows the constructed polygon. Here is the construction:

- insert unit lengths between each of the lengths of S; S' becomes $\{x_1, 1, x_2, 1, \ldots, 1, x_n\}$
- insert unit lengths to the beginning and end of S'; S' becomes $\{1, x_1, 1, x_2, 1, \ldots, 1, x_n, 1\}$
- add L to the end, where $L = \frac{1}{2}\sum_{i=1}^{n} x_i + 1$; S' becomes $\{1, x_1, 1, x_2, 1, \ldots, 1, x_n, 1, L\}$
- add v to the end, where $v = \mid S \mid + 1$; S' becomes $\{1, x_1, 1, x_2, 1, \ldots, 1, x_n, 1, L, v\}$
- add another L to the end; S' becomes $\{1, x_1, 1, x_2, 1, \ldots, 1, x_n, 1, L, v, L\}$

The result S' is an input instance to the 2D folding problem.

We now show that the Partition Problem reduces to the 2D folding problem by proving the following statement:

S can be partioned into two sets of equal sums $\Longleftrightarrow$ S' folds into an orthogonal polygon

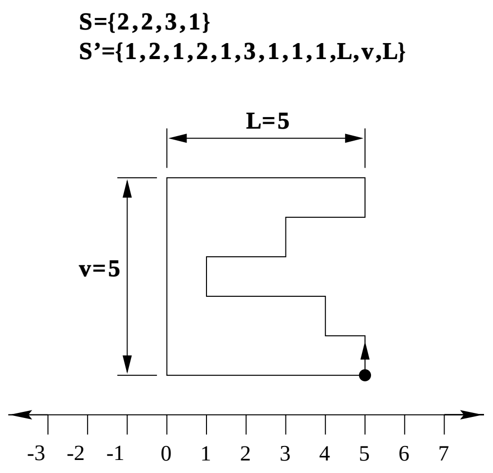


Figure 3.3: Constructing S' from the instance {2,2,3,1} of Partition.

where $S = \{x_1, x_2, \ldots, x_n\}$, $S' = \{1, x_1, 1, x_2, 1, \ldots, 1, x_n, 1, L, v, L\}$, $L = \frac{1}{2}\sum_{i=1}^{n} x_i + 1$, and $v = \mid S \mid +1$.

($\Rightarrow$) If the partition instance, S, has a solution $\{S_1, S_2\}$ where $\forall x_i \in S$, $x_i \in S_1$ or $x_i \in S_2$ and $\sum_{x \in S_1} x = \sum_{y \in S_2} y$, then the constructed 2D folding instance S' also has a solution. Assign a positive sign to all $x_i \in S_1$ and a negative sign to all $x_j \in S_2$. These will be horizontal edges. Each unit edge between the elements of S is assigned a negative sign. The L, v, L edges and the unit edges before and after the the elements of S together form C-clamp like chain. The result is a polygon. See the example in Figure 3.3.

($\Leftarrow$) Suppose the geometry instance, S' has a solution where

V_n = vertical lengths directed downwards,

V_p = vertical lengths directed upwards,

H_n = horizontal lengths to the left, and

H_p = horizontal lengths to the right.

We know that alternating elements of S' belong to V, where $V = V_n \cup V_p$ and the remaining elements belong to H, where $H = H_n \cup H_p$. Let the i^{th} (where i is odd) elements be in V and the j^{th} (where j is even) elements be in H. So V will be $\{1, 1, \ldots, 1, v\}$ and H will be $\{x_1, x_2, \ldots, x_n, L, L\}$. Since S' folds to a closed polygon by the statement of our implication, then both sets, V and H, must have a sub-partition such that the two subsets have equal sums. For V, this partition is trivial since the number of 1s is equal to v. So let V_n be v, and V_p be $\{1, 1, \ldots, 1\}$. We will now show that $\{x_1, x_2, \ldots, x_n\}$ has a partition. Recall that L is defined to be $\frac{1}{2}\sum_{i=1}^{n} x_i + 1$. Since $L + L = \sum_{i=1}^{n} x_i + 2 > x_1 + x_2 + \ldots + x_n$, the two Ls cannot be in the same subset, so each of H_n and H_p must contain exactly one L. If this is the case, then since the sums of the elements of H_n and H_p must be equal, then there must be a partition among $\{x_1, x_2, \ldots, x_n\}$.

The reduction can be computed in polynomial time. Since the Partition problem is NP-complete, it follows by the reduction that the 2D Folding Problem is also NP-complete.

□

3.2 NP-completeness of the 3D Problem

This section is the result of joint work with Therese Biedl.

Theorem 2: *The 3D folding problem is NP-complete.*

Proof: We show that the 3D problem is also NP-complete using the same approach as for the 2D problem. First, the 3D polyhedron folding problem is shown to be in NP. The input to the 3D folding problem is an orthogonal polygon with orthogonal undirected folds. The evidence used to verify that a polygon does fold into a polyhedron consists of the direction of each fold. With the input instance of a polygon and the directions given, we have exactly the input we are given to the problem presented in first section of this thesis. Using the method from the previous section, the polygon can be checked in polynomial time. Therefore, the 3D polyhedron folding problem is in NP.

A transformation is presented for constructing an input instance, S'', to the 3D folding problem from an input instance of the Partition Problem. The reduction of the Partition Problem to the 3D folding Problem is then shown. To illustrate the transformation, the diagrams throughout this section represent the example when S={1,2,1}, where S is an instance to the Partition Problem.

The following description shows how an input instance, S, to the Partition Problem can be transformed into S'', an input instance to the 3D folding problem.The first step in transforming S into an instance to the 3D folding problem is to construct a polygon exactly as we did for the transformation used in the NP-completeness proof of the 2D folding problem. That is followed by a series of steps to form an orthogonal polyhedron. The final step will be to describe an unfolding of this polyhedron – an orthogonal polygon which is S'' – the instance to the 3D folding problem corresponding to S.

Once the first step of transforming S to S' (the constructed polygon for the 2D folding problem) is complete, the resulting polygon is extruded to form a polyhedron (see Figure 3.4, step 2). The faces of this polyhedron that correspond to the edges of the 2D polygon, we call the *spine.* There are only two other faces of the polyhedron – the *front* and *back.* It is no good leaving the front and back as single faces in the unfolding, since their shapes would then give away, or force, the folding of the 2D polygon. Imagine instead an unfolding of this polyhedron, where the *front* and *back* faces are made from strips joined at the faces representing the vertical edges of the jagged edge of the polygon. If we are given the width of each of these strips, it forces the shape of the polygon. To avoid this, we replicate the jagged edge to the left of the vertical edge, so that the width of each strip is the same, regardless of the shape of the polygon (see Figure 3.4, step 3).

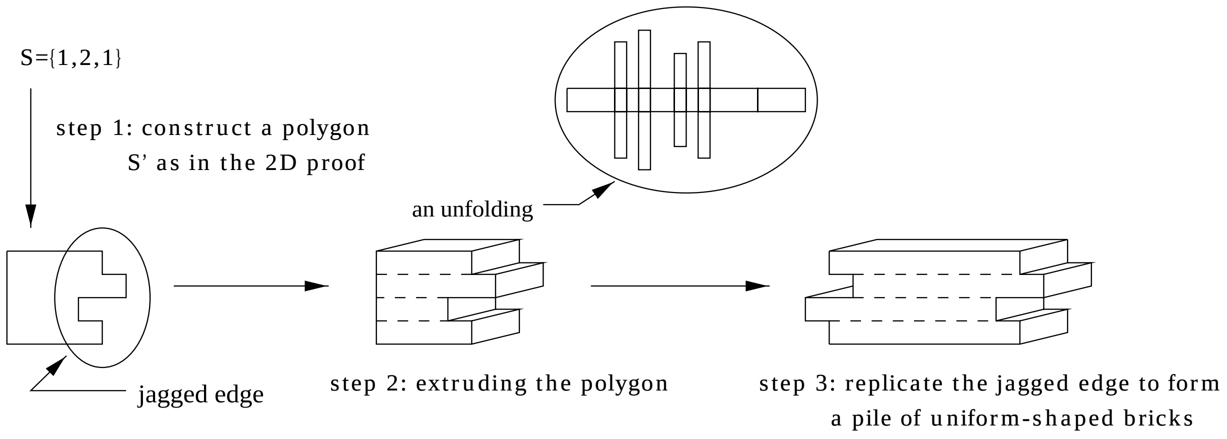

S={1,2,1}
step 1: construct a polygon S' as in the 2D proof
an unfolding
jagged edge
step 2: extruding the polygon
step 3: replicate the jagged edge to form a pile of uniform-shaped bricks

Figure 3.4: The first few steps of the construction

The length of each strip should be at least the sum of the integers plus one. This guarantees no intersection between the two jagged configurations. Although this solves the problem of not knowing the shape of the jagged edge, it inadvertently dissipates the requirement for the two vertical unit faces at the top and bottom of the jagged edge to be aligned along the same vertical axis. Hence, this no longer guarantees a solution to the partition problem of the original instance S. And so the transformed polygonal paper cutout for an instance of the partition problem with no solution will still fold into an orthogonal polyhedron. See the example in Figure 3.5.

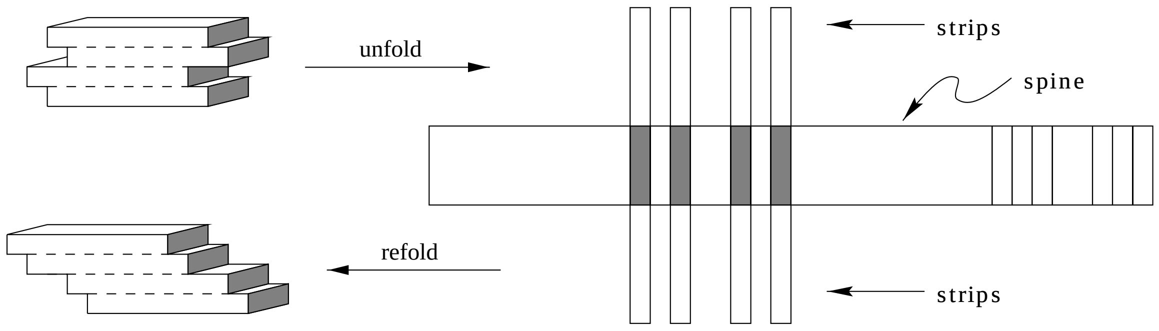


Figure 3.5: Step 3 does not guarantee the alignment of the top and bottom faces

To overcome this obstacle, we introduce a vertical crease along the front side of the polyhedron, and extrude the side of the polyhedron to the left of the vertical crease by a unit depth (see Figure 3.6, step 4). We call this crease the "shelf". Again, this shelf brings the problem of not knowing where to put folds for each strip covering the front faces since the shape of the jagged edge is not known. The final step in the transformation resolves this problem by scooping out square notches along the front face of the polyhedron (see Figure 3.6, step 5).

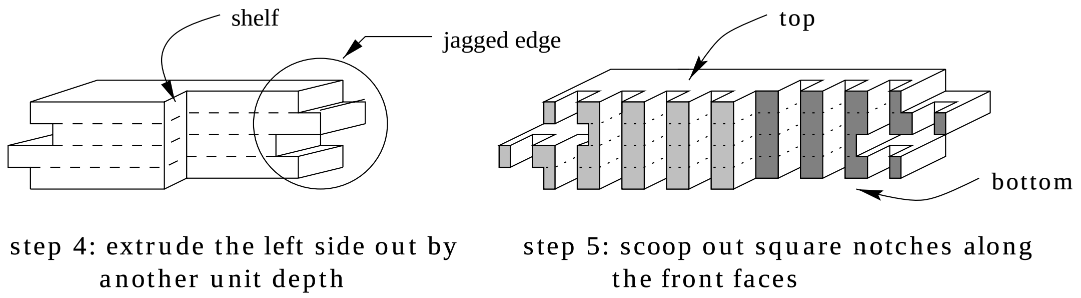

shelf
jagged edge
top
bottom
step 4: extrude the left side out by another unit depth
step 5: scoop out square notches along the front faces

Figure 3.6: The final steps of the construction

From the final polyhedron, we can easily deduce an unfolding that does not rely on knowing the shape of the jagged edge. We construct a band of connected polygons formed by all faces parallel to the xz-plane and the y-z plane with the exception of the shelf. The remaining faces to add will be the front and back faces plus the shelf. These faces will be constructed from strips of uniform length as mentioned before. The strips forming the front faces and the shelf will have many folds to accommodate the notches introduced in the last step of the transformation. An example of the orthogonal polygon generated for the instance S={1,2,1} to the partition problem is shown in Figure 3.7.

This completes the construction of an input instance, S", from an input instance to the Partition Problem, S'. We claim that,

S can be partioned into two sets of equal sums $\Longleftrightarrow$ S" folds into an orthogonal polyhedron

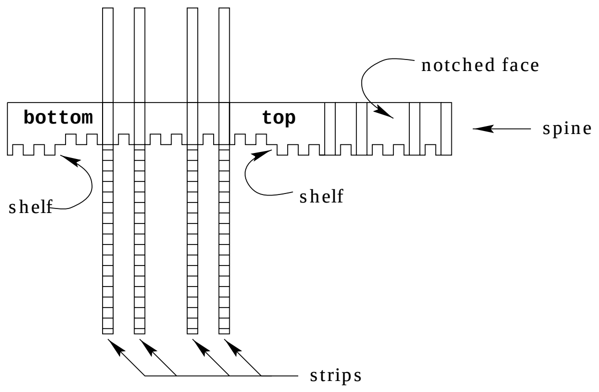


Figure 3.7: Polygon instance resulting from transformation from S={1,2,1}

Proof:

($\Rightarrow$) This direction of the claim is always true since the orthogonal polygon S'' is derived from unfolding a simple orthogonal polyhedron by the transformation described above.

($\Leftarrow$) This direction of the claim is not as straightforward. We need to show that the polygon, S'', cannot be folded into any other polyhedron but the one intended, in order to show that there must be a partition among edges corresponding to S. Orienting the axes as shown in Figure 3.8,

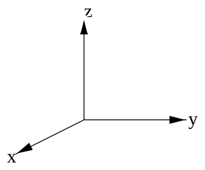
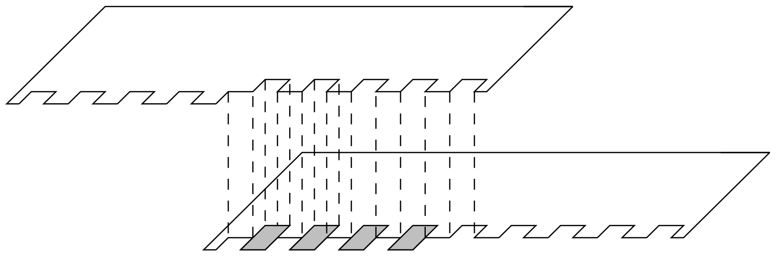


Figure 3.8: Orientation of axes

the faces of the spine all lie parallel to the xy-plane or the xz-plane once folded. The top and bottom faces lie perpendicular to the z axis. The faces of the spine must fold up to form (an extrusion of) a 2D polygon. The faces of the strips lie in the y-z and xz-planes. What prevents this polygon from looking like the lower left polyhedron in Figure 3.5? We need to show that the top face and the bottom face must line up – in particular, that the shelf in the top face must line up directly above the shelf of the bottom face. Suppose it doesn't, then you have something as in Figure 3.9.



The shaded area in the bottom face can be seen from above the top face.

Figure 3.9: When the top face does not line up directly above the bottom face

Consider the shaded area. Some face in the xy-plane must cover this area, so it cannot be a face of a strip. This leaves the faces of the spine to cover the *exposed* regions of the bottom face. Since the construction of the polygon makes the distance from the edge of the top and bottom faces to the shelf of these faces just long enough, the other faces of the spine are just not big enough to reach the shelf and back. Contradiction. So the top and bottom faces must line up. And hence the faces that form the jagged edge must begin and end at the same y coordinates. Using the same argument as in the 2D proof, it follows that S has a partition. Hence the 3D folding problem is NP-complete.

$\square$

Chapter 4

Folding Up the Polyhedron

Suppose we are given an orthogonal polygon with the indicated folds and we have determined that it does indeed fold into a simple polyhedron. We want to determine exactly how we should fold the cutout – do we do one fold at a time, or all folds at once?

If a polygon is made of some material that is malleable and gives in to pressure, like paper, and we are simply folding the polyhedron with our hands, it would seem that we can fold the polyhedron in any manner we wish. Suppose the polygon is made of some rigid material such as sheet metal and is folded using a machine that makes bends in the sheet metal. There are limits to what can be folded, and limitations to the order in which we make the bends in the cutout polygon. It is also undesirable to fold and then unfold, since the metal is weakened. Some questions that arise are, "Does the order in which we fold each fold matter?" "If we allow partial and simultaneous foldings of the folds, can any polygonal cutout be folded into its intended polyhedron?"

If a polyhedron and its cutout are fairly simple, then it doesn't really matter which order we make the folds. However, if the polyhedron to be constructed is rather complicated with many turns or twists and tunnels, some ordering of folds may require certain faces to *pass* *through* other faces, while another ordering of folds can avoid this from happening. The example in Figure 4.1 shows one order of folding that requires some parts of the polygon to pass through other parts of the polygon, and a second ordering of folds that does not.

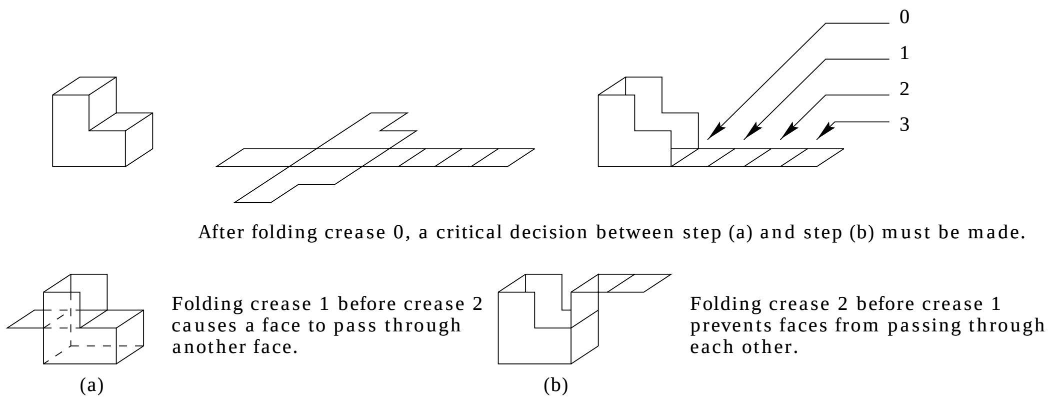

0
1
2
3
After folding crease 0, a critical decision between step (a) and step (b) must be made.
Folding crease 1 before crease 2 causes a face to pass through another face.
Folding crease 2 before crease 1 prevents faces from passing through each other.
(a)
(b)

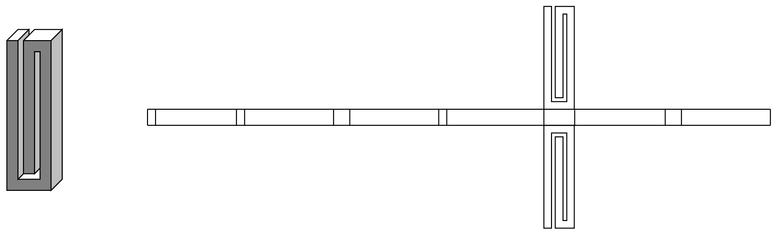
Figure 4.1: When ordering matters

Suppose we are limited to folding one fold at a time. Also, a fold must be complete (a full 90 degrees). That is, we cannot make folds simultaneously and we cannot fold a face half way, then continue with other folds, and return to the previous fold to complete it. And we do not allow any unfolding. Then the diagram in Figure 4.2 shows an extruded closed polygonal chain that cannot be folded from the particular cutout given this restriction.

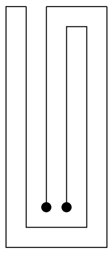
Claim 1: *The orthogonal polyhedron in Figure 4.2 cannot be folded from the rigid polygon shown if folds must be complete and folded one at a time.*

Proof: To fold the polychain, we must be able to fold the main strip of faces. If that can be achieved, the two faces covering the front and back faces can be folded up to complete the folding. However, we will show that with the given cutout, the main strip cannot be folded one complete fold at a time if the polygon is made of a rigid material.

We can view the faces in the strip and its folds as a sequence of line segments joined by hinges. Since the faces cannot cut through each other in the folding process, the problem



An example of a polyhedron and its net that cannot be folded one face at a time if the faces are rigid.



We can represent the main strip of faces as line segments and try to achieve the folding shown to the left. The last short segment to close the chain is ignored.

Figure 4.2: A polychain that cannot be folded one fold at a time

can be seen as a polygonal chain problem where the chain must be planar at every stage of the folding. The particular cutout shown corresponds to the polychain shown at the bottom of Figure 4.2.

Although it is not known if there exists a polychain which cannot be unfolded if joints can be unfolded simultaneously, it is well-known that some chains cannot be unfolded one joint at a time. For example, this fact is implied by work in progress by Arkin, Fekete, Mitchell, and Skiena [Dem99]. They prove that it is NP-hard to decide whether a chain can be unfolded one joint at a time. The polychain in Figure 4.2 is just one such polychain.

Clearly, if there is an unfolding of the polychain, then there is a folding of the straightened chain into the folded chain. To unfold the given polychain, one of the folds must be the first join to be straightened out. We can see that the end joints (the first ones from the endpoints of the chain) cannot be the first to be straightened out since the end segments are very long and are tucked inside. If we choose any other joint to be the first fold to be straightened out, all the other joints must stay in their fixed position at a right angle, but this will cause the rest of the polychain, including one of the end segments, to move simultaneously. Hence a contradiction since we have shown that neither of the end segments can be straightened out. Hence the polygon shown cannot be folded to form the polyhedron if folds are completed one at a time.

□

The polygon shown in Figure 4.2 does not fold into the polyhedron with the imposed restrictions, but it is easy to see there are other polyhedral nets of the polyhedron that can be folded with the same restrictions. The type of polyhedron in Figure 4.2 can be viewed as an extruded 2D polygon where two polygons of the same shape are joined by rectangular faces. The unfolding consists of the two main faces defining the outline of the polychain and a strip of rectangular faces which form the chain. If we remove the folding restriction and allow folds to bend simultaneously and at different rates, then there is no known extruded polygon that cannot be folded from an unfolding such as the one shown for the polychain.

But as we show now, there exists a cutout that cannot be folded if the faces are rigid, even if we allow simultaneous folding and partial folding.

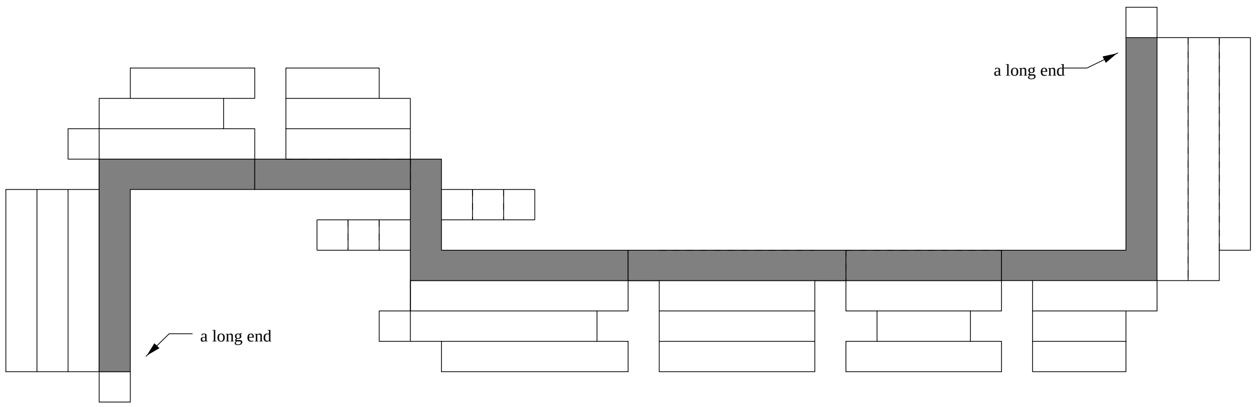


Figure 4.3: This net cannot be folded if its faces are rigid.

The cutout in Figure 4.3 folds to form the polyhedral knot shown in Figure 4.4. However, it cannot be folded if the ends are extremely long.

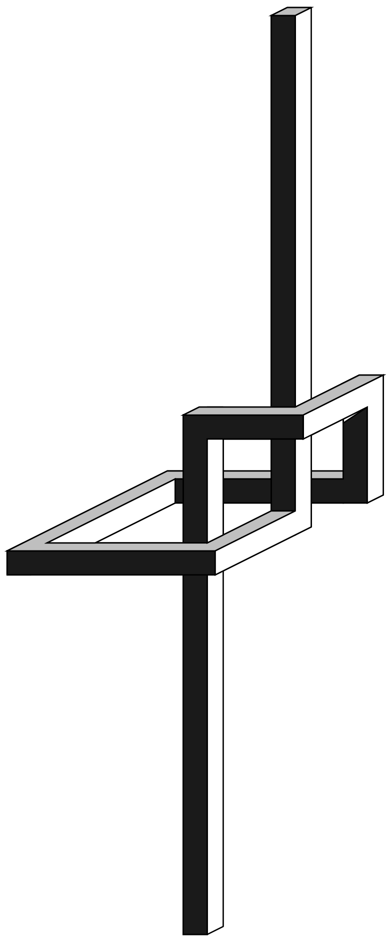


Figure 4.4: This "knot" cannot be folded if made from a rigid material like sheet metal.

Claim 2: *The cutout in Figure 4.3 cannot be folded to the polyhedron in Figure 4.4 if the faces are rigid.*

Proof: To show that the net given above cannot be folded into the orthogonal polyhedral trefoil knot, we will use the findings in [BDD$^+$99] which show that the chain in Figure 4.5a cannot be unlocked. Imagine we attach a thin iron rod to each face of the *spine* of the cutout (the spine is shown as shaded faces in Figure 4.3. And suppose each rod is joined to the next by some joint which allows the rods to bend only in the direction of the fold indicated (in this particular cutout, all folds are mountain folds). If we can somehow bend the iron rods in such a way that they form the chain as in Figure 4.5b, then we can see it may be possible to fold the cutout into the intended polyhedral "knot". This chain is similar to the one in Figure 4.5a except that in order to make it orthogonal, it must have seven smaller links instead of just three. By the same argument presented in [BDD+99], it is not possible to fold the rods that way. Although three of the smaller links in the cutout are a part of other links, this only restricts the movement of the link more. Therefore the cutout cannot be folded into the polyhedral knot.

□

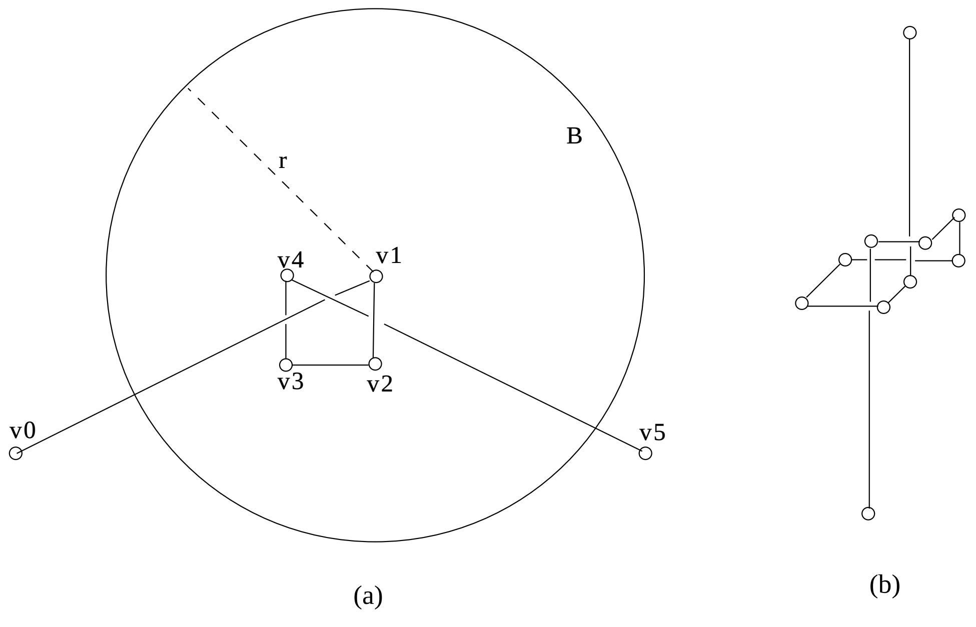


Figure 4.5: Knotted chains

Chapter 5

Conclusion

In this thesis, we defined orthogonal polyedra, and presented a problem of folding orthogonal polyhedra from orthogonal polygons. We gave an algorithm for determining whether a given polygonal cutout folds into a polyhedron.

In the second section, we presented the same problem with one less piece of input to the problem – the direction of folds. We showed that this problem is NP-complete. Our starting point was to prove that the two dimensional version of folding a linkage of orthogonal line segments into a polygon is NP-complete.

Although the problem of folding a polygon with undirected folds into a polyhedron is NP-complete, there are some polygons for which the direction of a fold is apparent. Such polygons are the ones that form extruded polygons where two faces defining the outline of the prism are connected to a main strip of parallelogram faces (e.g. the polychain in Fig. 4.2 and the L-shaped block in Fig. 4.1.)

We also saw that some cardboard cutouts can fold to form different polyhedra. We have seen a particular class of such polyhedra in the NP-completeness proof of the three dimensional folding problem. If we give the particular net for a polyhedron shaped as a pile of bricks, we have seen that this cutout can fold into different polyhedra determined by the directions of the edges that form a jagged edge.

The last section of the thesis focused on the task of actually folding a polygon into a polyhedron once it is determined that the folding does indeed form an orthogonal polyhedron. The actual folding of a polyhedron was subject to two restrictions (i) the faces of the cutout are rigid, and (ii) a fold must be complete (90 degrees) and all folded one at a time. We presented an instance of a polygon that could not be folded adhering to the two restrictions, and another instance that could not be folded with just the first restriction.

An extension to this problem is to find a characterization for a class of orthogonal polygons that can or cannot be folded given both restrictions. Once it is determined that a polygon can be folded given those restrictions, we could then find an algorithm that determines an ordering of the folds to be executed so that no faces need to pass through another.

Bibliography

[Ale58] A.D. Alexandrov. *Konvexe Polyeder.* Akademie-Verlag, 1958.

[BDD$^+$98] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *10th Canadian Conference on Computational Geometry*, pages 70–71, 1998.

[BDD$^+$99] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. Locked and unlocked polygonal chains in 3d. In *Proc. 10th ACM-SIAM Symposium. Discrete Algorithms*, pages 866–867, January 1999.

[Cox63] H.S.M. Coxeter. *Regular Polytopes.* The Macmillan Company, 1963.

[Cro97] Peter R. Cromwell. *Polyhedra.* Cambridge University Press, 1997.

[DDL$^+$99] E. Demaine, M. Demaine, A. Lubiw, J. O'Rourke, and I. Pashchenko. Metamorphosis of the cube. In *8th Annual Video Review of Computational Geometry, Proc. 15th Annual ACM Symposium on Computational Geometry*, 1999.

[Dem99] E. Demaine, March 1999. Personal communication.

[DK82] D.P. Dobkin and D.G. Kirkpatrick. Fast detection of polyhedral intersections. In *Automata, Languages and Programming, 9th Colloquium*, volume 140, pages 154–165. LNCS, 1982.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[GBKK98] S.K. Gupta, D.A. Bourne, K.H. Kim, and S.S. Krishnan. Automated process planning for robotic sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, 1979.

[Grü77] B. Grünbaum. Regular polyhedra, old and new. *Aequationes Mathematicae*, 16:1–20, 1977.

[HH88] I. Hargittai and M. Hargittai. Polyhedral molecular geometries. In *Shaping Space: A Polyhedral Approach*, pages 172–188. Design Science Collection, 1988.

[LO96] A. Lubiw and J. O'Rourke. When can a polygon fold to a polytope? Technical Report 048, Smith College, June 1996.

[RW91] L. Roth and G.L. Wybenga. *The Packaging Designer's Book of Patterns*. Van Nostrand Reinhold, 1991.

[Sen88] M. Senechal. Introduction to polyhedron theory. In M. Senechal and G. Fleck, editors, *Shaping Space: A Polyhedral Approach*, pages 191–197. Birkhäuser Boston, Inc, 1988.

[WW86] P. Widmayer and D. Wood. A time- and space-optimal algorithm for boolean mask operations for orthogonal polygons. Technical Report CS-86-39, University of Waterloo, Department of Computer Science, 1986.